VŠB - Technická univerzite Ostrava

Fakulta elektrotechniky a informatiky

DIPLOMOVÁ PRÁCE

2009 **Petr Kalafatič**

VŠB - Technická univerzite Ostrava

Fakulta Elektrotechniky a Informatiky

Implementace BitTorrent protokolu

Implementation of BitTorrent protocol

2009 **Petr Kalafatič**

Prohlášení

Prohlašuji, že diplomovou práci jsem vypracoval samostatně.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne …… ……..
podpis

Poděkování

Rád bych poděkoval Ing. Michalu Radeckému za jeho rady a věcné připomínky.

Anotace diplomové práce

Název práce : Implementace BitTorrent protokolu

Autor : Petr Kalafatič

Obor : Informatika a výpočetní technika
Druh práce : Diplomová práce

Vedoucí práce : Ing. Michal Radecký
Fakulta elektrotechniky a informatiky, VŠB – Technická univerzita Ostrava

Abstrakt :
Diplomová práce obsahuje popis implementace BitTorrent protokolu. Práce je rozdělena na pět částí, z nichž první část se zabývá popisem P2P sítí a konkrétněji BitTorrent protokolem. Druhá část se zabývá návrhem systému - obsahuje návrh architektury a použité struktury. Třetí část se věnuje popisu použitých technologií.Čtvrtá část se věnuje vlastní implementaci systému. Poslední část tvoří závěr a zhodnocení.

Klíčová slova :
Torrent, tracker, seed, peer, swarm, funkční modul, komponenta, plugin.

Další použité výrazy a zkratky popsány na konci textu ve slovníku pojmů.

Annotation of Thesis

Title : Implementation of BitTorrent Protocol

Author : Petr Kalafatič

Field of study : Informatics and Computer Technology
Kind of study : Diploma Thesis

Person in authority : Ing. Michal Radecký, Department of Informatics,
College of Electrotechnics and Informatics VŠB – Technická Univerzita Ostrava

Abstract :
This Thesis is proposed to implement BitTorrent Protocol. This Thesis is divided into five parts. The first part presents P2P networks and the BitTorrent protocol, second part presents design of the system; contains architecture design and structures. The third part contains description of used technologies. The fourth part presents description of implementation and last part contains summary

Key words :
Torrent, tracker, seed, peer, swarm, functional modul, component, plugin.

More keywords in dictionary at the end of Thesis.

Obsah

1. Úvod

Při výběru tématu diplomové práce jsem využil možnosti zvolit si téma, které mě zajímá. Jedná se o problematiku datových přenosů, výměnných sítí - konkrétně P2P(viz slovník pojmů) sítí. V zimním semestru roku 2006 jsem v rámci předmětu Internetové technologie zpracovával volnou implementaci BitTorrent protokolu. Po zkušenostech z této úlohy jsem se rozhodl vytvořit plnohodnotnou aplikaci, která by byla použitelná v praxi. Hlavní myšlenkou projektu je vytvořit komplexní platformu pro podporu datových přenosů na bázi BitTorrent protokolu. Nejedná se tedy pouze
o vlastní datové přenosy, ale i o podporu těchto činností. Je to například vyhledávání torrent souborů, možnost kontroly dat (přehrávání mediálních souborů, kontrola sítě, rozšíření protokolu o novou funkcionalitu atd.). Současné implementace jsou více či méně zaměřeny pouze
na stahování dat.

Inspirací a výzvou pro mě byla aplikace Vuze[16] (BitTorrent klient), o které je nutné se zmínit, aby bylo jasné, jak vypadá současný špičkový BitTorrent klient. V čem je ale ona inspirace ?

Vuze mě na první pohled zaujal díky přístupu k uživateli, kdy ovládání programu a nastavování vlastností (často poměrně složitých) je intuitivní a přehledné. Taktéž lze vyzdvihnout stabilitu programu. Pokud již dojde k nějaké výjimečné situaci (například výpadek připojení k internetu), program se sám zotaví z této chyby a pokračuje v činnosti. Většina těchto situací je uživateli také vysvětlena na informačních panelech, které se v těchto situacích zobrazují.

V současné době jde Vuze[16] cestou poskytování rozmanitých služeb, především pak sdílení videa, což je díky „nativnímu“ vysokokapacitnímu BitTorrent protokolu a obrazu o vysokém rozlišení téměř revolučním řešením pro budoucí využití internetu.

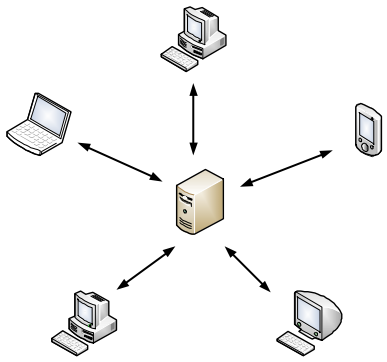
Obrázek 1 - Hlavní okno BitTorrent aplikace Vuze[16]

Vuze[16] je napsaný v Javě pod licencí GPL(viz slovník pojmů) tudíž je možno nahlédnout pod pokličku perfektně zvládnuté aplikace. Od verze 2.3 využívá DHT (distribuovaná databáze- zvyšuje odolnost proti výpadkům serveru), rozeznává lokalizaci (překlad textových částí programu) atd.

2. Základní informace

2.1. Problematika přenosů dat

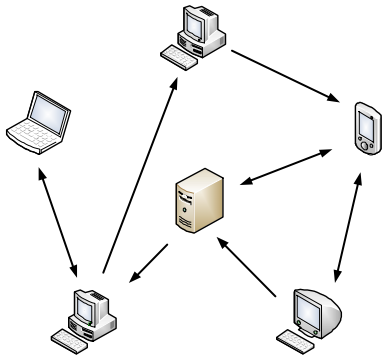
U klasických (klient/server) aplikací (na obrázku 2), které v minulosti převládali, je největší problém v tom, že klienti požadují po serveru data v podstatě sekvenčním způsobem. To znamená, že se klient zařadí do fronty a čeká, až na něj přijde řada (nebo modifikace tohoto řešení se stejnými výsledky). Klient je závislý na možnostech serveru, počtu ostatních klientů, velikosti dat atd. U takového systému na první pohled vidíme co nám vadí – jsou to fronty čekajících klientů.



Obrázek 2 – Klient/Server síť

Jedna z možností řešení těchto problémů je zapojit klienty, tak, aby se využil jejich potenciál. Myšlenkou je využít server pouze jako zprostředkovatele („moderátora“). Tento server pouze předává informace, kde hledat data. Ve výsledku to znamená, že čím větší je zájem (fronta klientů), tím je systém efektivnější (v porovnání s klient/server řešením).

Výše popsaná možnost řešení je zjednodušenou definicí BitTorrent protokolu[11], který je tématem této práce. Obrázek 3 pak naznačuje, jak tato síť vypadá.



Obrázek 3 - BitTorrent síť

Autorem BitTorrent protokolu je americký programátor Bram Cohen[12] (narozen 1975). Od pěti let se učil programovat v Basicu, účastnil se matematických olympiád. V roce 1993 dokončil střední školu (Stuyvesant High School), která je zaměřená především na matematiku. Poté pokračoval ve studiu na vysoké škole v Buffalu, kterou ovšem nedokončil a dále se již věnoval práci pro několik společností. Podstatnou inspirací byl systém MojoNation, kde se v podstatě rozdělený soubor distribuuje na počítače a uživatelé si je takto po kouskách stahují. Další motivací byla dlouhá doba stahování velkých souborů ze serverů. Cohen navrhl BitTorrent protokol tak, aby bylo možné data stáhnout z různých zdrojů, s ohledem na potenciály jednotlivých klientů (jako je například přenosová rychlost připojení k síti). BitTorrent protokol vytvořil v roce 2001 a uveden byl v roce 2002. Bram Cohen[12] v současnosti pracuje na vyhledávacím systému, který bude shromažďovat odkazy vedoucí k tisícům programů, filmů a dalších aplikací sdílených pomocí systému BitTorrent.

Obrázek 4 - Bram Cohen

2.2. Popis BitTorrent protokolu

BitTorrent protokol se ve stručnosti dá popsat jako soubor pravidel pro přenos dat vzájemně propojených počítačů (v dnešní době bychom již neměli uvažovat o počítači jen jako o kusu hardwaru pod stolem, ale i o mobilních telefonech, kapesních počítačích a jiných zařízeních). Cohen přišel s nápadem rozdělit přenášený soubor na menší části, které každý ze stahujících uživatelů může okamžitě odesílat dalším uživatelům ve swarmu (uživatelé stahující jeden soubor). Tímto způsobem dojde k rovnoměrnému rozdělení souboru mezi všechny peery (uživatel, který stahuje soubor) a vzájemnou výměnou dojde jednak ke stažení kompletního souboru a k rozmělnění zatížení sítě.

Protokol komunikace dle specifikace se skládá v principu ze 2 částí (kompletní specifikace BitTorrent protokolu je v příloze). V první části se uvádějí pravidla pro komunikaci mezi klientskou aplikací a trackerem (server, který zprostředkovává komunikaci mezi klienty). Klientská aplikace odesílá údaje, které obsahují identifikaci klienta, identifikaci torrentu a dále data, která jsou nutná pro správný chod sítě. Klient posílá trackeru svůj identifikátor (20 bytový řetězec) a port na kterém aplikace naslouchá (typicky rozsah 6881-6889). Následuje hash (viz slovník pojmů), který specifikuje torrent a další údaje, jako kolik má klient již staženo, kolik adres klientů požaduje
a jiné.Ve druhé části se uvádí pravidla pro komunikaci mezi klienty. Tato část obsahuje především handshake, který také obsahuje specifikace torrentu a identifikátor klienta. Následují pravidla pro výměnu dat a potřebných informací, která udávají jaká data a od koho tyto data stahovat (podrobný popis je rozebrán dále).

Pokud bychom chtěli analyzovat problémy (nevýhody) systému, pak musíme lehce přijít k otázkám typu :

1. Co se stane, když bude tracker nedostupný ?
2. Klienti budou nedostupní ?
3. Nebude k dispozici celá kopie (bude chybět kus dat) ?
4. Jakým způsobem ovlivňuje BitTorrent aplikace síťové zatížení ?

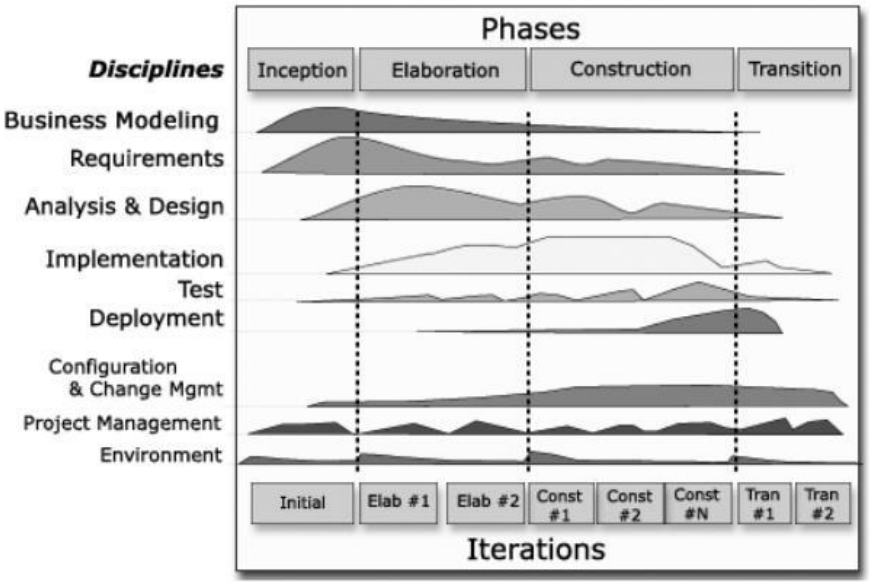
Tyto otázky se snaží řešit některá mladší řešení. Jde především o problém decentralizace sítě tak, aby jednotlivé klientské aplikace nebyly závislé na nějakém centrálním (statickém) serveru. Jako příklad dva systémy:

1. eXeem – hybrid BitTorrent + FastTrack, není tak úspěšný, jak se čekalo
    a. není závislý na souborech typu .torrent, ale na koordinujících serverech (což ale není příliš přínosné)
    b. na rozdíl od BitTorrent protokolu, který je pod licencí GPL (viz Slovník pojmů) je eXeem komerční aplikace
    c. FastTrack nepřichází s ničím novým
2. Kademlia (DHT[1]) – decentralizovaná síť, kalkulace metrik vzdáleností uzlů a klientů uvnitř sítě
    a. není závislý na souborech typu .torrent, ale stejně se musí nějakým způsobem připojit do sítě – k tomu může využít seznam uzlů z internetu (což také není ideální)
    b. je šířena pod licencí GPL a bohužel bylo vytvořeno několik jejích vzájemně nekompatibilních modifikací

2.3. Vývoj aplikace

Protože u tak velkých projektů je téměř nutností plánovat jednotlivé kroky dlouho dopředu, je u tohoto projektu snaha využít stávající osvědčené metodiky, konkrétně RUP [14]. RUP[14] je propracovanou objektově orientovanou iterativní metodologií vývoje softwaru, která obsahuje čtyři základní části, přičemž každá z nich je organizovaná do několika iterací. Za iterace v tomto projektu můžeme považovat vývoj jednotlivých funkčních modulů, či konzultace s vedoucím práce.

Fáze zahájení byla v tomto případě v podstatě výběrem smysluplného tématu diplomové práce, které, jak již bylo uvedeno, bylo „prozkoumáno“ několik měsíců předem v rámci semestrálního projektu. Další fází je projektování, kde byly analyzovány potřeby projektu, potencionálních uživatelů a definovány základy architektury. Bylo využito statistických údajů z průzkumu, který byl zaměřen na průzkum preferencí týkajících se softwaru pro sdílení dat potencionálních uživatelů (kompletní výsledky a zdroje statistiky jsou součástí přílohy diplomové práce). Třetí fází je realizace, která byla zaměřena na vývoj návrhu aplikace a zdrojových kódů. V poslední fázi předání byla již pouze laděna stávající implementace (tedy práce na vývoji nových komponent či funkcí byla zastavena a důraz byl kladen na upevnění stability stávajícího stavu projektu).



Obrázek 5 – RUP – schéma [15]

Obrázek 5 ilustruje rozložení činností během vývoje softwaru dle RUP[15] tak, jak to bylo popsáno výše.

3. Návrh a architektura systému

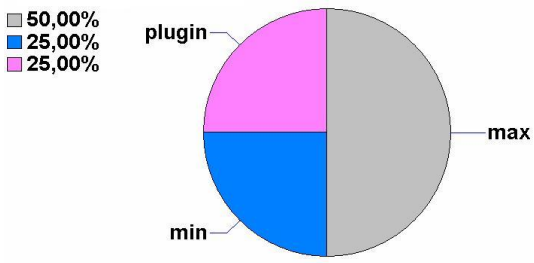
3.1. Rozbor statistického průzkumu

Cílem statistického průzkumu bylo zpřesnit specifikaci požadavků, v závislosti na výsledcích analyzovaných dat. Statistický průzkum vychází z dotazníkového šetření (prováděného v období od 10.4. do 25.4.2008). Tento způsob získávání požadavků (pomocí dotazníků) je standardním nástrojem UML[16] (jazyk pro modelování systémů) . Kompletní dokumentace a zdrojové soubory jsou k dispozici v příloze na CD.

Mezi nejdůležitější otázky dotazníku z pohledu návrhu aplikace patří:

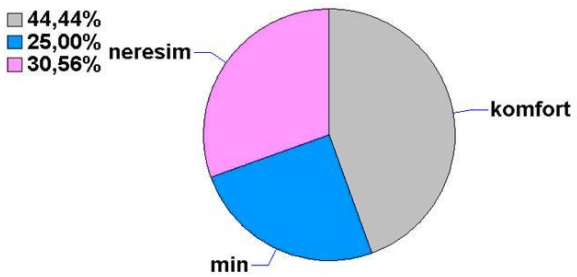
- Jaká struktura SW (software) vám vyhovuje ?
- Kolik funkcí by BitTorrent klient měl mít ?
- Vaše nároky na GUI (grafické rozhraní) ?

U první otázky šlo v podstatě o získání přehledu, jak by měla vypadat struktura aplikace. Buď je program kompletní – jen aplikaci nainstaluji nebo je minimální - poskytuje jen minimální konfiguraci, tak aby pouze splnil základní funkce nebo třetí varianta - poskytuje možnost sestavení, jako stavebnice, kdy si poskládám aplikaci podle sebe. 50% respondentů chce vše v jednom balíku. Čtvrtinu zajímá pouze holá funkčnost a čtvrtina chce mít možnost sestavit aplikaci dle svých potřeb.



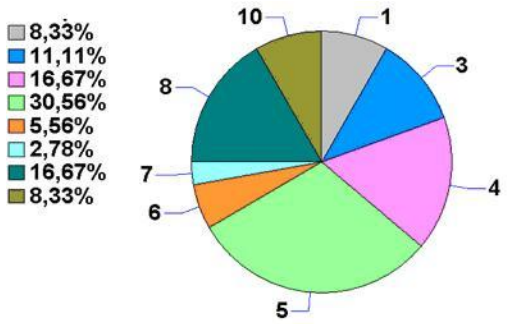
Obrázek 6 – Koláčový graf výsledku dotazu na strukturu programu

Následující charakteristika. 44% respondentů požaduje komfortní funkčnost, ovládání maximálního počtu funkcí, nastavení charakteristik aplikace k obrazu svému.
Zhruba třetina naopak požaduje minimum funkcí, pro jednoduché ovládání (z vlastní zkušenosti vím, že někdy se opravdu uživatelé ztrácejí v možnostech programu, což může vést k zatracení jinak třeba výborného programu). Třetina respondentů pak funkce neřeší.

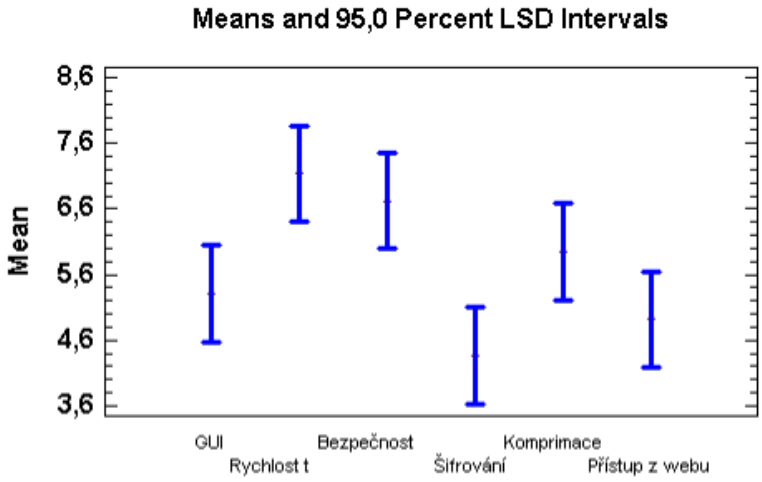


Obrázek 7 - Koláčový graf výsledku dotazu na funkčnost programu

Zhruba třetině respondentů na vzhledu příliš nezáleží, třetina dala pod 5 a třetina nad 5 bodů preference. U téhle charakteristiky bych byl trochu opatrný vzhledem k povaze programátorů (kteří byly respondenty) -“Není až tak důležité, jak to vypadá, hlavně, že to funguje”.
V reálném světě bude zřejmě zjev hrát podstatnější úlohu.



Obrázek 8 - Koláčový graf výsledku dotazu na vzhled programu



Obrázek 9 - Graf souvislostí mezi hodnocením některých funkcí

3.2. Specifikace požadavků

Požadavky lze definovat jako specifikaci toho, co by mělo být implementováno.

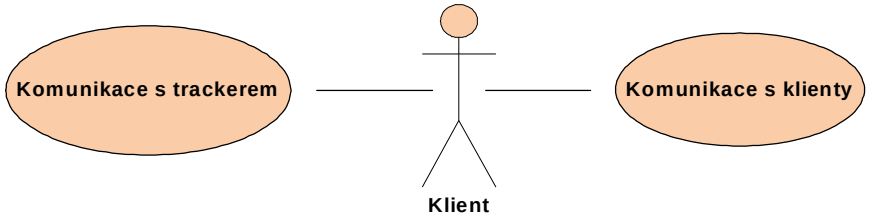
1. Funkční požadavky:
   - 1.1. Funkční požadavky na aplikaci
     - 1.1.1. Spuštění aplikace
       Spuštění aplikace je spojené se zobrazením loga a průběhu spouštění
       Při prvním spuštění zobrazení uvítacího okna s možností seznámení se s aplikací
     - 1.1.2. Hlavní menu
       Hlavní menu se skládá z položek : File, Edit, Tool, Project, Window, Help
       Do těchto základních položek pak přispívají doplňky svými podpoložkami
     - 1.1.3. Restart aplikace
       Uživatel má možnost restartu aplikace z hlavního menu
     - 1.1.4. Ukončení aplikace
       Uživatel má možnost ukončení aplikace z hlavního menu
     - 1.1.5. Zobrazení nápovědy
       Nápověda popisující funkce programu bude dostupná z hlavního menu
     - 1.1.6. Zobrazení informací o konfiguraci aplikace
       Uživatel má přístup k informacím o konfiguraci z hlavního menu
     - 1.1.7. Nastavení parametrů
       Uživatel má možnost změnit základní nastavení aplikace
     - 1.1.8. Kontakt na podporu projektu
       Uživatel má možnost kontaktovat vývojáře (chyby v programu, diskuse)
     - 1.1.9. Aktualizace
       Uživatel má možnost aktualizovat konfiguraci vybráním položky z hlavního menu
   - 1.2. Funkční požadavky na klientskou část aplikace
     - 1.2.1. Načtení/odstranění torrentu
     - 1.2.2. Ovládání stahování torrentu (spuštění, pozastavení, zastavení stahování/sdílení)
     - 1.2.3. Zobrazení stavu stahování

2. Nefunkční požadavky:
   - 2.1. Aplikace bude napsána v jazyce Java
   - 2.2. Aplikace využije Eclipse [2] RCP [5] technologii
   - 2.3. Aplikace využije EMF [6] technologii

3. Technické požadavky:
   - 3.1. Aplikace bude připravena pro běh pod operačním systémem Windows
   - 3.2. Aplikace bude distribuována v podobě komprimovaného souboru typu zip

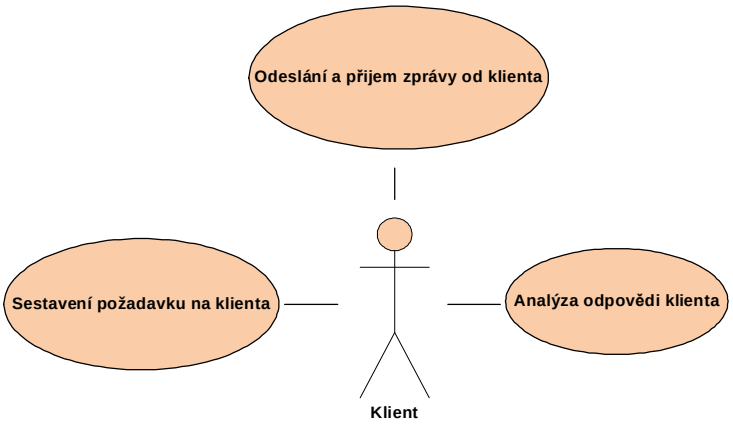
3.3. Případy užití

Základem většiny síťových systémů je komunikace mezi aplikacemi formou danou protokolem. V případě BitTorrent protokolu tento definuje v podstatě dva typy komunikace:



Obrázek 10 – Případ užití 1: BitTorrent komunikace

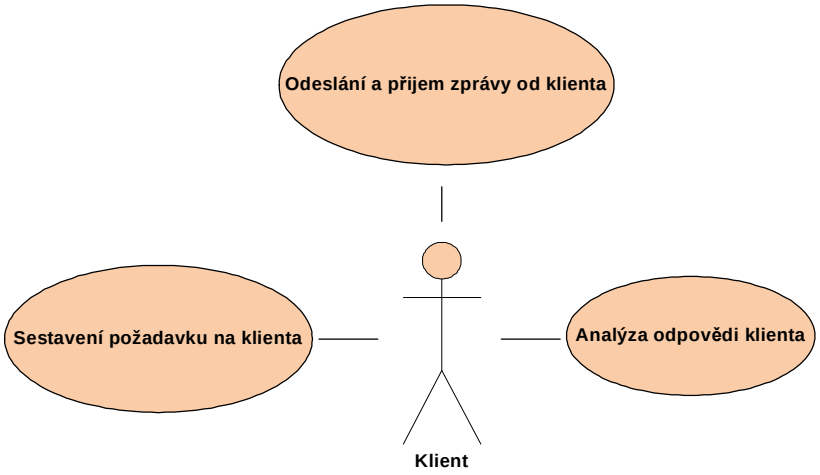
Obrázek 10 znázorňuje celkový pohled na případ užití BitTorrent aplikace.
Tento jednoduchý případ užití navazuje na předchozí kapitolu 2.2, kde byla popsána základní struktura komunikace v rámci BitTorrent protokolu.



Obrázek 11 – Případ užití 2: Komunikace klient/tracker

Kroky při komunikaci mezi klientem a trackerem:

1. Systém sestaví požadavek na tracker, dle aktuálního stavu
2. Systém pošle trackeru požadavek a přijímá odpověď
3. Systém vyhodnotí odpověď trackeru



Obrázek 12 – Případ užití 3: Komunikace klient/klient

Kroky při komunikaci mezi klienty:
1. Systém sestaví požadavek na klienta, dle aktuálního stavu
2. Systém pošle klienta požadavek a přijme odpověď
3. Systém vyhodnotí odpověď klienta

Tracker – Tok informací

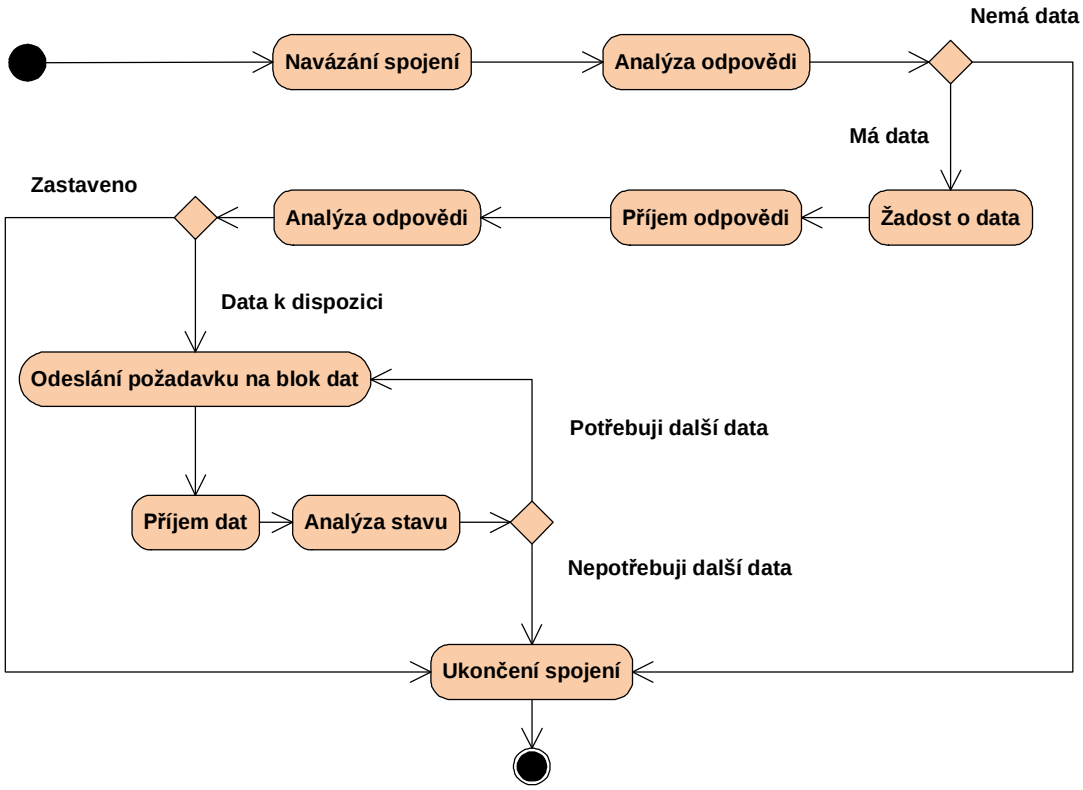
Tracker je jednoduchý server, který má za úkol poskytovat informace o klientech, kteří jsou v jednom swarmu (v rámci jednoho torrentu).
Po přijetí požadavku, je tento vyhodnocen, a když tracker má tento torrent v databázi (měl by mít, protože je tento tracker primárně uveden v torrent souboru), odešle zpět požadovaná data (adresy klientů ve swarmu).

Obrázek 13 –Diagram Aktivit 1: Tracker – komunikace s klientem

Tento diagram aktivit je protějškem komunikace klient/tracker, která byly znázorněná v případu užití 2. V rámci aktivity Analýza požadavků tracker rovněž aktualizuje svá data (databázi klientů), což je už ale detailnější pohled.

Klient – Stahování dat

Klient v první řadě navazuje spojení s klientem, od kterého chce data stahovat. Způsob tohoto navázání spojení je pevně určen protokolem a obsahuje v podstatě základní informace (viz následující kapitola 4.1 o kódování). Následuje komunikace, jejímž cílem je dohodnout se, která data jsou předmětem zájmu. Pokud klient tyto data má k dispozici, jsou požadavky konkretizovány (především upřesnění velikosti přenášených dat), a je započata fáze přenosu dat. Stažená data jsou analyzována a porovnávána (obecně se jeden soubor stahuje z několika zdrojů). Ukončení spojení nastává v případech, kdy již další data nejsou třeba nebo nejsou k dispozici.



Obrázek 14 – Diagram Aktivit 2: Klient – stahování dat

Klient – Odesílání dat

Klient má standardně spuštěný server, který poslouchá na defaultním portu (6881).
Tento server přijímá spojení a ověřuje, zda jsou požadavky splnitelné. Pokud má klient data torrentu k dispozici, dojde k předání informací o podmínkách přenosu a tyto data jsou postupně odesílána.

4. Použité technologie a struktury

4.1. BitTorrent – kódování

Implementace BitTorrent protokolu vyžaduje korektní manipulaci se zprávami, které jsou přenášeny po síti. Dostupné specifikace jsou poměrně kvalitně zpracovány, nicméně jejich implementace a především vyladění v ostrém provozu je časově velmi náročné.

Kódování je v podstatě předpis, podle kterého probíhá komunikace a přenos dat mezi dvěma koncovými body. Definuje pravidla řídící syntaxi, sémantiku a synchronizaci vzájemné komunikace. Laicky řečeno je to vytvoření komunikačního jazyka. Například informace „kladné celé číslo 123“ je v BitTorrent protokolu kódováno i123e.

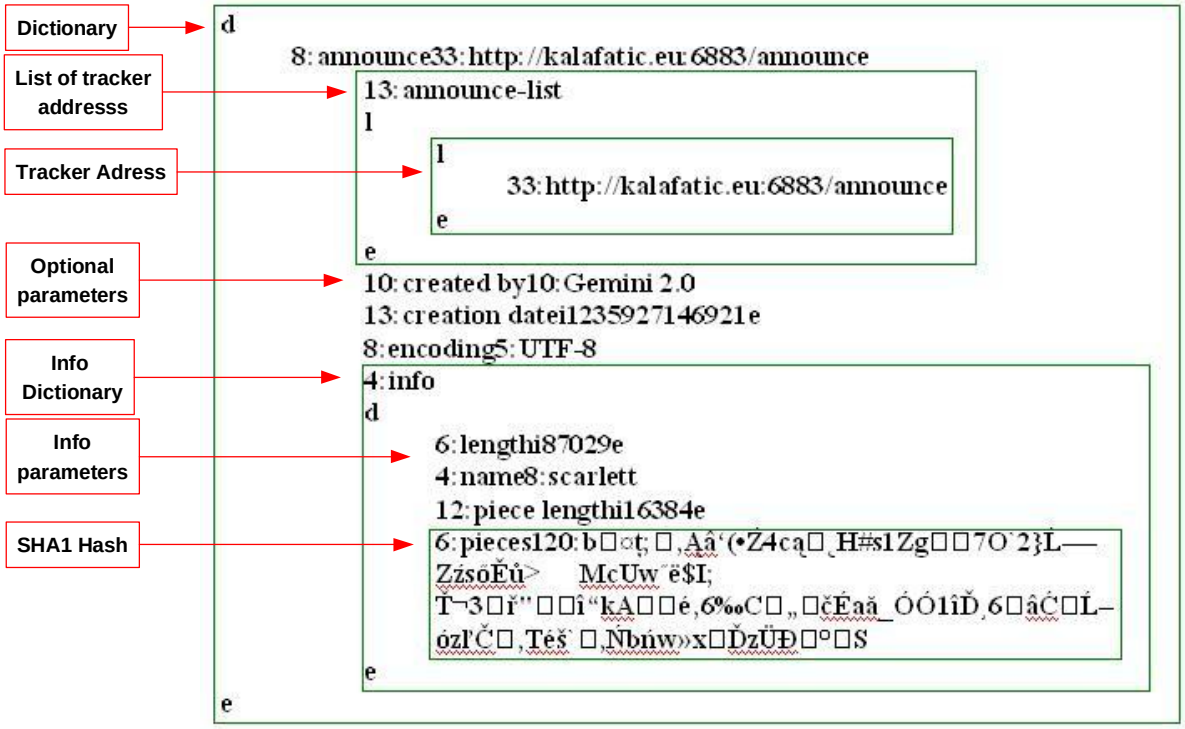
Torrent kódování :

Typ	Pravidlo kódování	Příklad
String	**délka stringu : string**	**6:string**
Integer	**i integer e**	**i123e**
List	**l položky e**	**l6:stringe**
Dictionary	**d string e**	**d7:string17:string2e**

Torrent metainfo struktura :

Typ	Pravidlo kódování	Poznámka
announce	**string**	**adresa trackeru**
announce-list	**string**	**seznam adres trackerů/ nepovinné**
creation date	**string**	**v sekundách/ nepovinné**
created by	**string**	**id klienta/ nepovinné**
encoding	**string**	**nepovinné**
comment	**string**	**nepovinné**
info		
private	**integer**	**0-1/ nepovinné**
pieces	**string**	**20 bytový SHA1 hash / pro každý díl**
piece length	**integer**	**délka dílu/násobky 2/obvykle kolem 512**
1 soubor		
-name	**string**	**jméno souboru**
-length	**integer**	**délka souboru**
-md5sum	**32 hex string**	**nepovinné**
více souborů		
-name	**string**	**jméno souboru**
-files		**seznam souborů**
-length	**integer**	**délka souboru**
-md5sum	**32 hex string**	**nepovinné**
-path	**string list**	**cesta k souboru**

Torrent soubor má v podstatě stromovou strukturu. Struktura začíná slovníkem, pokračuje seznamem adres trackerů a ostatních atributů. Ukázka jednoduchého torrentu vygenerovaného z obrázku typu JPEG (pro názornost odsazeno a zvýrazněno) je na následujícím obrázku 15.



Obrázek 15 – Struktura souboru .torrent

K obrázku 15 je vhodné dodat, že slovník „info“ obsahuje položku „pieces“, která se dá popsat následovně :

- je to řetězec délky 20*n, kde n je počet bloků, na které je sdílený obsah rozdělen;
- každá část tohoto řetězce délky 20 obsahuje hash kód n-tého bloku dat.

V ukázkovém případě tedy vidíme, že :

- velikost souboru je 87029 bytů
- blok má definovánu velikost 16384 (2^14), tedy 87029/16384=5,31 -> zaokrouhlíme nahoru na 6 bloků.
- pro každý blok vypočteme 20-bytový hash -> pieces=120 bytů.

Pokud by se zdrojová data skládala z několika souborů, pak by torrent soubor obsahoval klíčové slovo „files“, který by byl seznamem souborů podobně jako adresy trackerů v seznamu adres trackerů.

Klient – Tracker HTTP (protokol pro přenos hypertextových dokumentů) kódování požadavku/odpovědi

Typ	Pravidlo kódování	Poznámka
GET		
info_hash	20 bytový hash	Hash bloku „info“ z torrent souboru
peer_id	20 bytový string	-GE2010-000000000000
port	integer	Port, na kterém naslouchá klient
uploaded	integer	Počet odeslaných bytů
downloaded	integer	Počet přijatých bytů
left	integer	Počet bytů, které zbývá stáhnout
compact	0/1	1 znamená 6 bytový string (4 IP+2 port)
no_peer_id	0/1	Vynechání peer_id
event		Nepovinná položka udávající událost
-started	string	Začátek stahování
-stopped	string	Přerušení stahování
-completed	string	Dokončené stahování
ip		IP adresa klienta
numwant	integer	Počet požadovaných adres / Nepovinné
Key	string	Nepovinné
tracker id	string	Nepovinné
POST		
failure reason	string	
warning message	string	
interval	integer	Čas (v sekundách), po kterém by měl klient vyslat další dotaz na tracker
min interval	integer	Nepovinné
tracker id	string	Nepovinné
complete	integer	Počet seedů
incomplete	integer	Počet peerů
peers		Seznam slovníků s položkami
-peer id	20 bytový string	
-ip	string	
-port	integer	

Klient – Klient kódování „handshake“

Typ	Pravidlo kódování	Příklad
pstrlen	byte	19
pstr	string	BitTorrent protocol
reserved	8 bytů	00000000
info_hash	20 bytů	20bytový hash kód klíče info souboru metadat
peer_id	20 bytový string	-GE2010-000000000000

Klient – Klient kódování zpráv

Typ	Pravidlo kódování	Poznámka
keep alive	<len=0000>	Udržuje spojení
choke	<len=0001><id=0>	Oznamuje změnu příznaků spojení
unchoke	<len=0001><id=1>	Oznamuje změnu příznaků spojení
interested	<len=0001><id=2>	Oznamuje změnu příznaků spojení
not interested	<len=0001><id=3>	Oznamuje změnu příznaků spojení
have	<len=0005><id=4><piece index>	Potvrzuje přijetí bloku, následuje číslo bloku
bitfield	<len=0001+X><id=5><bitfield>	Posíláno pouze jednou při navazování spojení
request	<len=0013><id=6><index><begin><length>	Žádost o data
piece	<len=0009+X><id=7><index><begin><block>	Samotná data
cancel	<len=0013><id=8><index><begin><length>	V ukončovacím módu stahování
port	<len=0003><id=9><listen-port>	

Praktická ukázka implementace kódování zpráv :

```
private static final byte[] KEEP_ALIVE = { 0x00, 0x00, 0x00, 0x00 };
private static final byte[] CHOKE = { 0x00, 0x00, 0x00, 0x01, 0x00 };
private static final byte[] UNCHOKE = { 0x00, 0x00, 0x00, 0x01, 0x01 };
private static final byte[] INTERESTED = { 0x00, 0x00, 0x00, 0x01, 0x02 };
private static final byte[] NOT_INTERESTED = { 0x00, 0x00, 0x00, 0x01, 0x03 };
private static final byte[] HAVE = { 0x00, 0x00, 0x00, 0x05, 0x04 };
private static final byte[] PORT = { 0x00, 0x00, 0x00, 0x03, 0x09 };
```

Zajímavé je kódování „bitfield“, proto následuje detailnější rozbor. Zpráva bitfield udává, které části sdíleného souboru má klient k dispozici. Soubor se rozdělí na díly (rozdělení souboru na díly je dáno velikostí souboru a torrentem definovanou délkou dílu). Tyto díly se deklarují jako 0/1 (0-klient nemá danou částici, 1-klient má danou částici). Tyto díly se kódují jako bity, přičemž chybějící bity jsou nastaveny na 0.

Příklad:

Mějme soubor rozdělený na 10 částí,přičemž klient již stáhl 2. díl:

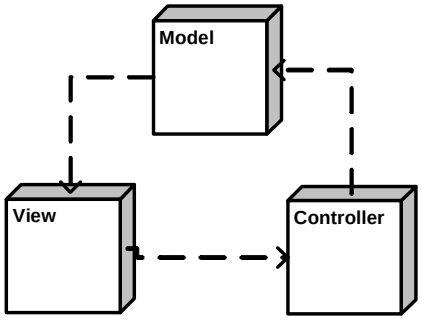
- **01000000 00**000000 (soubor je vyjádřen bitovou reprezentací bytu zleva doprava)
- Bitfield budou tvořit 2 byty: 64 0
- Pokud klient stáhne 1. díl:
- **11000000 00**000000
- Bitfield budou tvořit 2 byty: -64 0
  (Byty jsou znaménkové (-128... 127), tudíž 11000000 se kóduje -64 a ne 192)

4.2. Technologie a struktury

Použité technologie a struktury jsou dány jednak nefunkčními požadavky specifikace požadavků, dále pak jejich použití vyplývá z potřeb projektu (ukládání a zobrazování informací a dalších). Využití některých složitějších technologií (viz dále) přináší efektivnější a rychlejší vývoj aplikací. Nicméně na druhou stranu je nutné věnovat poměrně dost času jejich nastudování.

MVC[1] (Model View Controller)

Základní myšlenkou architektury MVC je oddělení uživatelského rozhraní a logiky aplikace. V tomto projektu je tento princip implementován takto:
okenní panel zobrazuje data z modelu a nabízí akce pro jejich modifikace. Akce jsou vykonávány kontrolerem, který mění data v modelu. V rámci kontroleru jsou na model navázáni posluchači, kteří po změně modelu aktualizují zobrazení (tabulky, stromy). Výhodou tohoto přístupu je strukturalizace kódu, což ve výsledku vede k požadované přehlednosti a udržení konzistence kódu, což je neocenitelné u projektů, které se budou vyvíjet delší dobu a je předpoklad, že na něm budou pracovat i jiní programátoři.



Obrázek 16 – MVC[1] architektura

Eclipse RCP[5] (Rich Client Platform)

Eclipse RCP je open-source projekt, který nabízí možnost vyvíjet "vizuálně bohaté" desktopové aplikace v Javě. Představuje minimální množství pluginů (minimální konfiguraci), které je nutné k inicializaci prostředí. Jeho posláním je zjednodušit a zrychlit vývoj komplexních desktopových aplikací. (V podstatě nám nabízí IDE Eclipse k vlastnímu použití).
Aplikace jsou díky Javě platformě nezávislé a díky SWT[9] mají na každé platformě nativní vzhled. Eclipse RCP[5] je součástí rodiny open-source projektů Eclipse Project, které jsou vyvíjeny pod vedením IBM. Eclipse RCP[5] Runtime Component Model je založen na OSGi (Open Services Gateway Iniciative) a Extension/Extension Point modelu. V jednoduchosti to znamená, že je definován standard, který umožňuje přidávání dalších částí do původní aplikace, přičemž tyto části spolu komunikují na základě daného rozhraní (Extension Point).

EMF[6] (Eclipse Model Framework)

EMF[6] je framework pro modelování a generování Java kódu. Jeho nejdůležitější vlastností (díky které je použit v této práci) je vizuální zpracování návrhu architektury modelu, generování kódu a diagramů a funkce, které jsou k dispozici pro manipulaci s modelem. V průběhu návrhu architektury byla zvažována možnost využít pro editaci dat některou z klasických databází.
Pro použití EMF[6] rozhodly tyto důvody :

1. velikost manipulovaných dat není velká a nejsou mezi nimi složité vazby
2. vizualizace struktury dat a z toho vyplývající snadnější udržení konzistence dat
3. předpoklad (splněný), že podpora EMF[6] bude na standardně vysoké úrovni
4. ukládání dat ve formě XML(viz slovník pojmů) v sobě má určitý potenciál (transformace do jiného typu dokumentu, výměna dat)

Adapter

EMF[6] nabízí možnost, jak zachytit změnu objektu (nejedná se tedy o návrhový vzor), a to dvěma způsoby :

1. definujeme třídu rozšiřující AdapterImpl a implementující Adapter (instance takového objektu se používá pro „hrubé“ změny (například přidání nebo odebrání objektu z kolekce))
2. definujeme třídu rozšiřující EContentAdapter a implementující Adapter (instance takového objektu se používá pro zachycení změny atributu objektu (nastavení))

Grafická knihovna SWT [9]

V prvních distribucích Javy bylo obsaženo GUI AWT (Abstrakt Windows Toolkit), které však nebylo příliš dokonalé. Další GUI se kterým SUN přišel byl Swing. Swing má již dostatek dobrých vlastností, nicméně byl trochu obětován výkon. SWT (Standard Widget Toolkit) je grafická knihovna pro Javu, která nemá závislost na standardním grafickém API. Výhodou SWT je fakt, že je svázán s nativními službami, což přináší rychlost a vzhled přebírá z příslušného OS. Nutno zmínit, že nadstavbou SWT je framework JFace, který poskytuje množství „High Level“ komponent, které se používají právě v Eclipse RCP[5].

JSF (Java Server Faces)

Technologie byla vyvinuta společností Sun Microsystems, Inc. Je součástí Java 5 Enterprise Edition. Vývojáři, kteří používají tuto technologii definují uživatelské rozhraní pomocí speciálních XML(viz slovník pojmů) tagů, kterým jsou předávána data k zobrazení nebo editaci ze standardních Java Beans. Takto je rozdělena webové aplikace čistě na uživatelské rozhraní a aplikační logiku.

V rámci webové podpory je tato technologie použita pro tvorbu webových stránek projektu. Tyto stránky jsou propojeny s databází MySQL. Pro tvorbu stránek bylo využito vývojové prostředí NetBeans, které podporuje MySQL databázi a webový server Apache Tomcat, takže v podstatě odpadají nepříjemnosti způsobené rozmístěním stránek a databáze na serveru.

MySQL[1]

MySQL[1] je multiplatformní databáze, která je snadno implementovatelná (lze jej instalovat na Linux, MS Windows, ale i další operační systémy), má dobrý výkon a jedná se volně šiřitelný software.

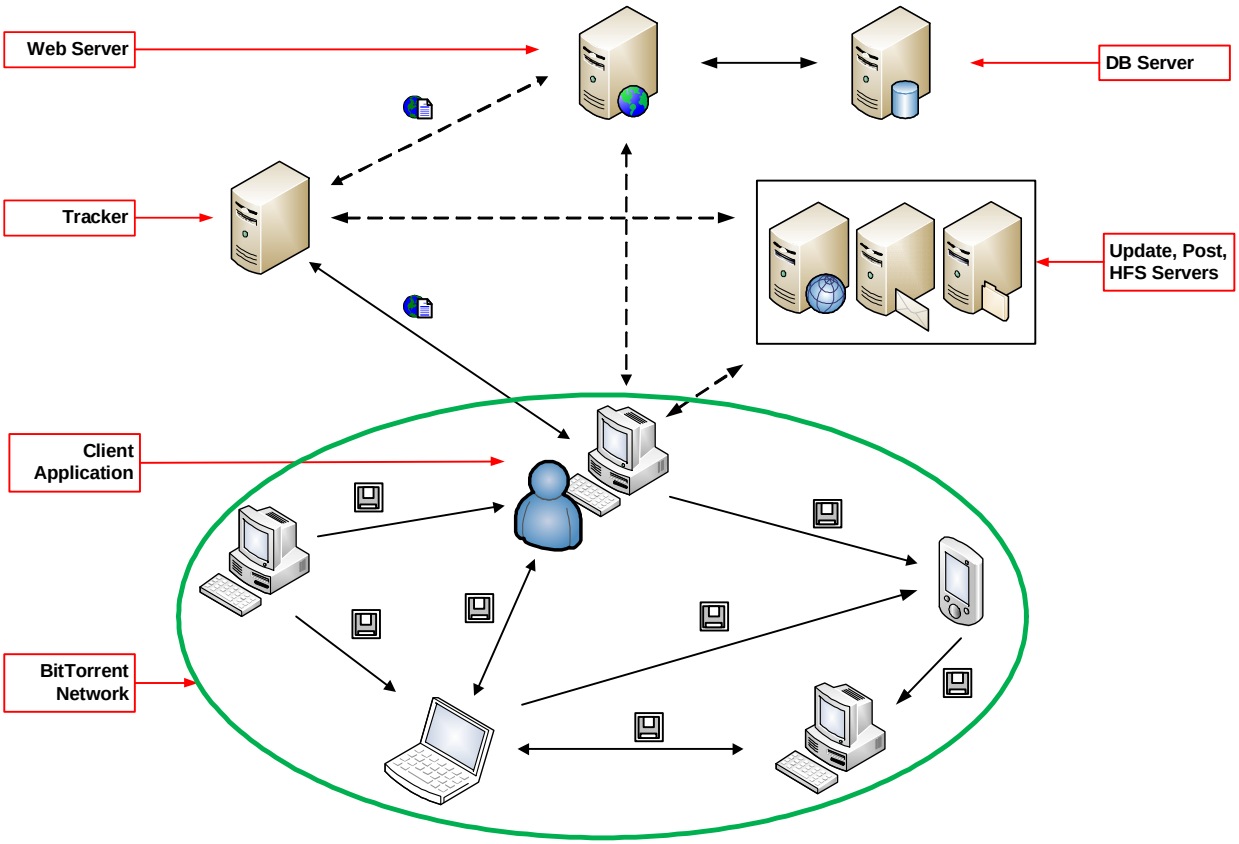
V rámci webové podpory je tato databáze použita hlavně proto, že je to jednodušší databáze a má kvalitní napojení na další prvky (Java, PHP, webový server Apache).

5. GNU Gemini

5.1. Motivace

Projekt Gemini je implementací BitTorrent protokolu, napsanou v programovacím jazyce Java. Základem je RCP[5] platforma, která je opravdu "bohatá" z hlediska využití nepřeberného množství doplňků a funkcionality. Hlavním cílem projektu bylo vytvořit robustní a komplexní aplikaci (platformu), která by snesla pozdější velké úpravy (přidávání / odebírání / aktualizace) komponent.

Systém je vyvíjen dle RUP[14] a velký důraz je kladen na přísné dodržení strukturalizace v rámci MVC, což samozřejmě přináší hlavně přehlednost kódu, udržení konzistence atd.
Gemini z těchto zdrojů čerpá.



Obrázek 17 – Kompletní systém Gemini projektu

Z tohoto obrázku (17) je patrné, jakým způsobem projekt pokrývá celou životní etapu torrentu. Pomocí aplikace je torrent vytvořen, může být vystaven na veřejném webovém serveru a stažen s pomocí taktéž veřejně přístupného trackeru. Dále projekt obsahuje kostru základní webové podpory, která již neoddiskutovatelně patří mezi standardní vybavení softwaru 21. století.

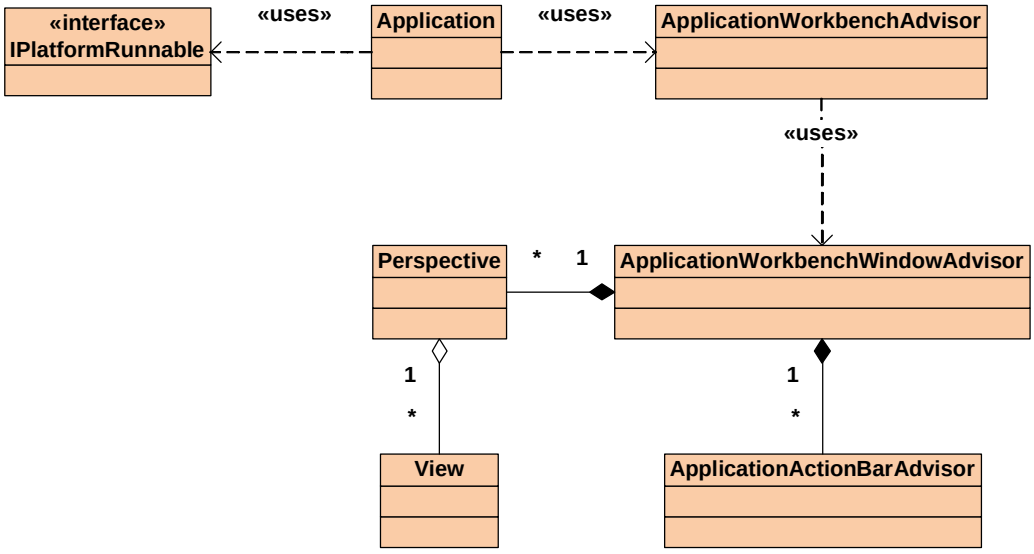
5.2. Jádro

Prvním implementačním krokem po návrhu architektury bylo vytvořit jádro aplikace, v našem případě RCP[3] aplikaci. Jádro má za úkol vytvořit prostředí pro další pluginy (je to jednoduše řečeno rámec*(framework)* budoucí aplikace). V rámci jádra jsou definovány obecné akce (aktualizace, nápověda ..) plus podporu přidávání (*contributors*) dalších akcí pro další pluginy (akce v menu ..) pro všechny funkční moduly. Jádro taktéž obsahuje definici produktu, která využívána pro ladění všech pluginů v rámci projektu (sledování závislostí, spolupráce pluginů funkčních modulů), informace potřebné pro sestavení, atd.
Jádro je ve struktuře pojmenováno jako *gemini.core*.

Základními třídami, které RCP[3] aplikace obsahuje jsou :

1. Application – třída zodpovědná za běh celé aplikace. Zavádí aplikaci (bootstrap) a otevře hlavní okno (Workbench Window)
2. ApplicationActionBarAdvisor – třída zodpovědná za vytvoření menu (Menu Bar, Tool Bar, Status Line)
3. ApplicationWorkbenchAdvisor – podporuje přidávání funkcionalit v rámci životního cyklu aplikace (například umožňuje přidat akce po startu nebo při ukončování aplikace)
4. ApplicationWorkbenchWindowAdvisor – jako u předcházející třídy – umožňuje přidávat akce, inicializuje základní parametry jako například rozměry hlavního okna
5. Perspective – třída, která definuje uživatelské rozhraní obsahující okna, která mají dohromady poskytovat informace k jedné funkcionalitě
6. View – okno, které má zobrazit konkrétní informace (například adresářovou strukturu)

Tyto třídy jsou pak vzájemně propojeny vazbami zobrazenými třídním diagramem na následujícím obrázku 18.



Obrázek 18 – Diagram tříd: RCP[3] aplikace

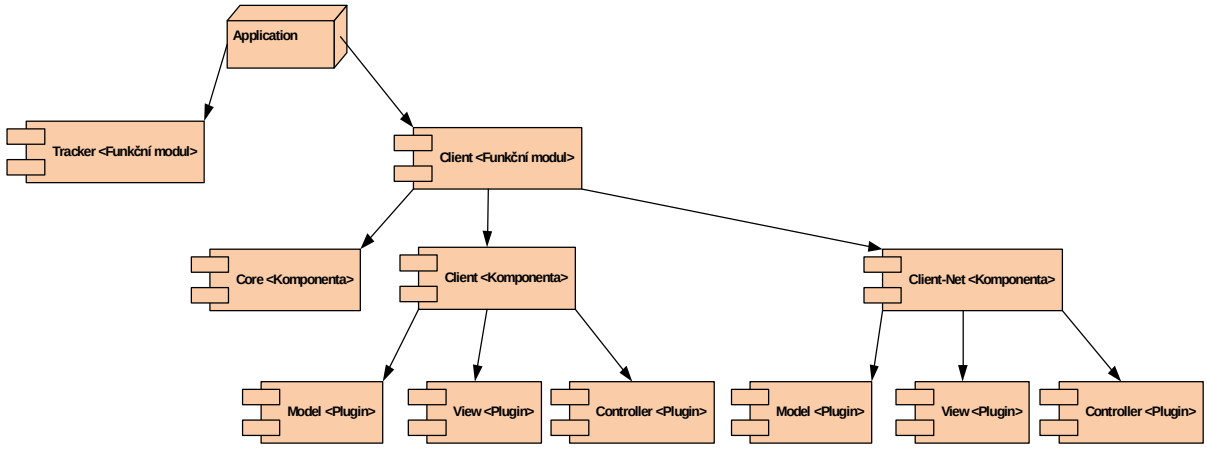
Následující ukázka zdrojového kódu ukazuje jednu z nejdůležitějších tříd RCP[3] aplikace. Konkrétně se jedná o třídu Application, která je zodpovědná za vytvoření hlavního okna aplikace.

```
public class Application implements IApplication {

    // ----------------------------------------------------------

    public Object start(IApplicationContext context) {

        Display display = PlatformUI.createDisplay();

        try {
            int returnCode = PlatformUI.createAndRunWorkbench(display,
                    new ApplicationWorkbenchAdvisor());

            if (returnCode == PlatformUI.RETURN_RESTART) {
                return IApplication.EXIT_RESTART;
            }
            return IApplication.EXIT_OK;
        } finally {
            display.dispose();
        }
    }

    // ----------------------------------------------------------

    public void stop() {
        final IWorkbench workbench = PlatformUI.getWorkbench();
        if (workbench == null)
            return;
        final Display display = workbench.getDisplay();

        display.syncExec(new Runnable() {
            public void run() {
                if (!display.isDisposed())
                    workbench.close();
            }
        });
    }
}
```

Obrázek 19 – Ukázka kódu: Třída zodpovědná za vytvoření hlavního okna aplikace

Tato třída je ekvivalentem metody main() běžné Java aplikace.

Struktura programu je důležitým prvkem aplikace-je ovlivněna použitou technologií (na bázi pluginů); nejmenším stavebním kamenem je proto z tohoto hlediska plugin. Protože je velikost projektu a doba jeho vývoje náročná z pohledu udržení konzistence, je dalším prvkem struktury větší funkční jednotka postavena na bázi MVC, a to komponenta, která obsahuje pluginy pro model, zobrazení a kontroler. Několik takových komponent, pokrývajících jednu funkcionalitu, je součástí funkčního modulu. Je to účelné i z důvodů podpory aktualizací v platformě RCP[3] a jiných, již zmiňovaných výhod. Toto pojmenování je použito pro potřeby popisu v tomto textu, v RCP[3] projektu jde pouze o rozdělení na *feature* (funkční modul), a *plugin* (komponenta tedy slouží jen jako zjemnění popisu struktury projektu). Tyto prvky pak tvoří celý systém.



Obrázek 20 – Detail MVC architektury Gemini

Poznámka : Obrázek 20 není kompletním diagramem projektu, je to jen část pro ilustraci, jakým způsobem je projekt členěn.

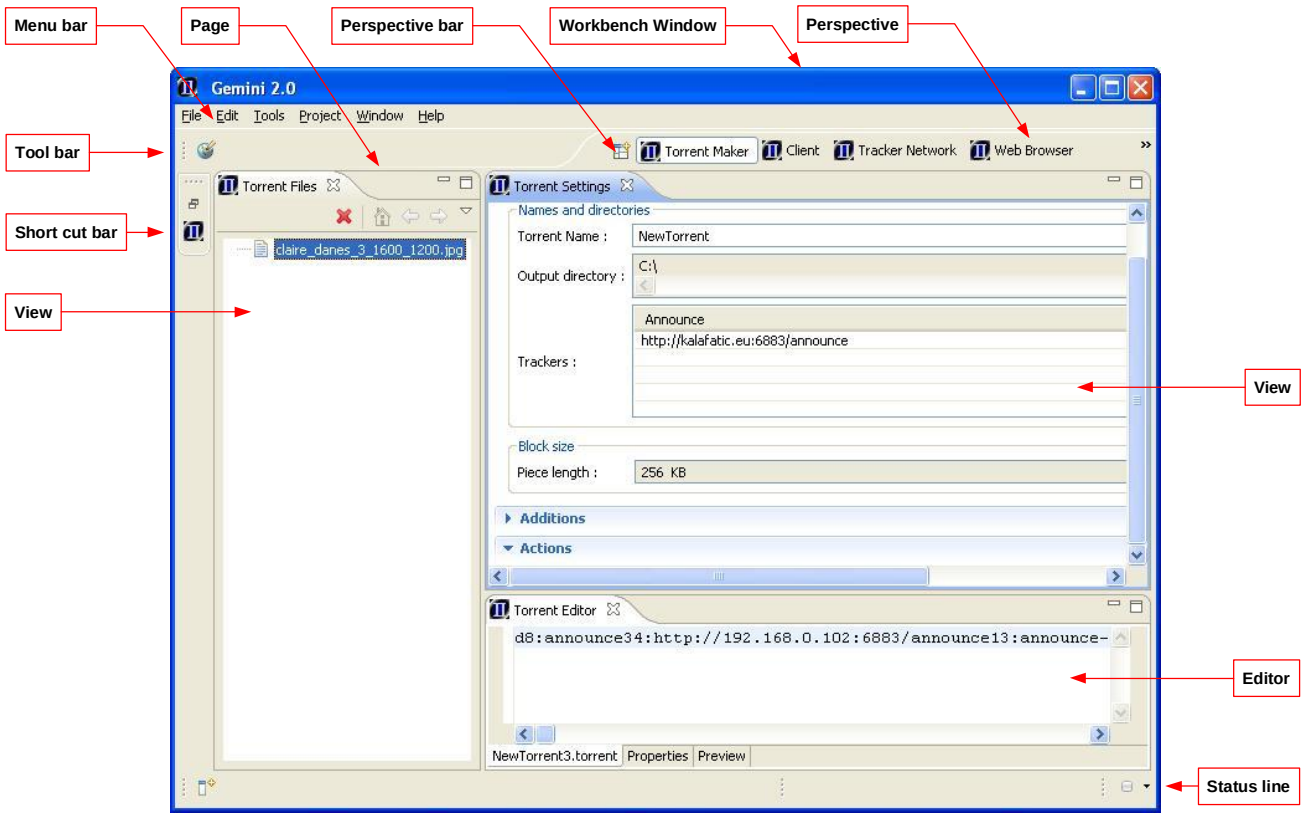
Plugin jako nejmenší část diagramu na obrázku 20 obsahuje především soubor pro definici pluginu typu XML(viz slovník pojmů) a balíky obsahující jednotlivé třídy. Jmenná konvence balíků je organizována dle typů a funkcí tříd, které obsahuje. Například třídy rozšiřující vlákna (Thread) jsou v jednom balíku. Orientace je tedy poměrně přehledná a rychlá.

Po vytvoření jádra aplikace nastal čas pro bližší pohled na grafické rozhraní RCP[5] aplikace. Struktura jádra (viz obrázek 18) obsahuje mimo jiné i třídy pro správu grafického rozhraní (*advisors*). Ty jsou užitečné hlavně proto, že definují přístupové body pro jiné pluginy.

Příklad:
V jádře je definováno standardní hlavní menu (File, Edit..). Funkční modul pro tvorbu torrentů jednoduše přidá svojí položku (File/Create Torrent) tak, že v deklaraci příslušné akce nastaví parametr MenuBar path.

Výčet nejdůležitějších prvků pojmů, které se v rozhraní RCP[5] aplikace nachází:

- Workbench – reprezentuje uživatelské rozhraní samotné a má jedno nebo více pracovních oken (Workbench Window)
- Workbench Window – hlavní okno Eclipse, může jich být více (další Window otevřeme z hlavního menu). Tato funkčnost má své uplatnění pro použití více monitorů (což je třeba při vývoji tohoto projektu neocenitelné). Každé takové okno má své menu a další různé prvky.
- Workbench Page – pracovní plocha okna Workbench – obsahuje editory a další okna (Views).
- Site – Objekt pomocí něhož dochází ke komunikaci mezi editory a okny.
- Perspective – toto je nástroj, který obsahuje libovolné editory a okna, která jsou k dispozici. Uživatel má možnost si vytvořit vlastní perspektivu, dát do ní okna, která potřebuje, rozmístit je a přiřadit jim velikost.
- Editor – vlastní komponenta sloužící k editaci
- View – okna ve Workbench Part – dají se rozmisťovat po pracovní ploše v rámci perspektivy.



Obrázek 21 – Rozhraní RCP[3] aplikace: demonstrace na perspektivě Torrent Maker

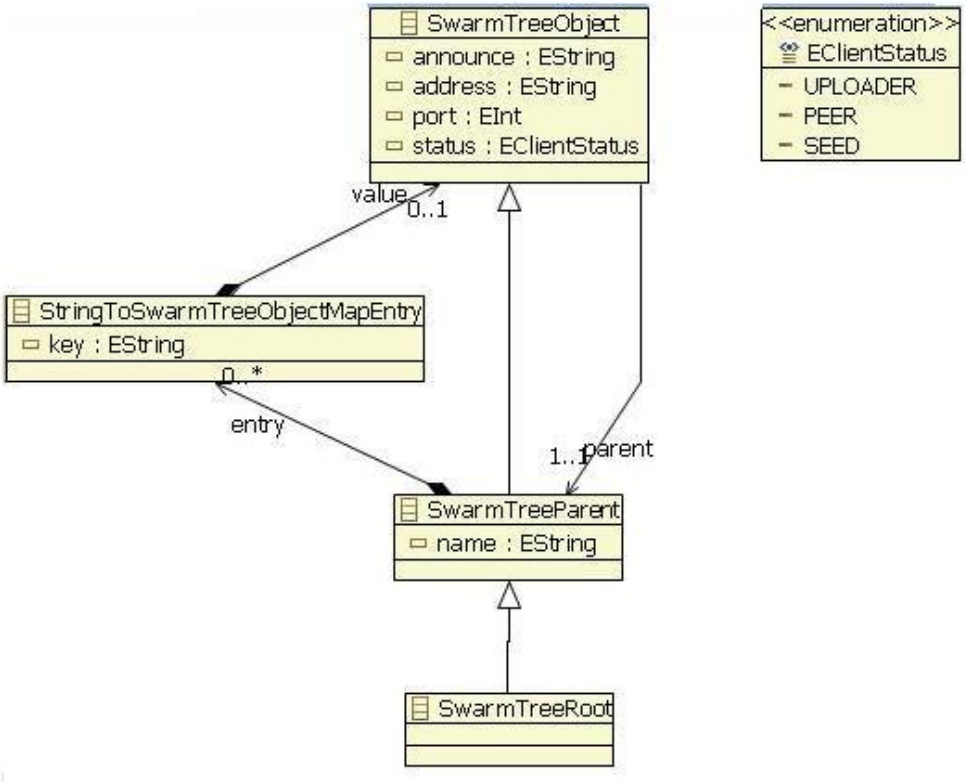
5.3. Klientská aplikace

5.3.1. Modely a jejich vizualizace

Klientská komponenta má za cíl především řídit činnosti spjaté se stahováním (odesíláním) dat, popsaných příslušnými torrenty. Struktura klientské aplikace je dána požadovanou funkcionalitou a pojetím architektury. Klientská aplikace obsahuje tři základní komponenty: *gemini.core, gemini.core.client* a *gemini.core.client.net*.

Klientská aplikace obsahuje především modely (EMF[6]) *gemini.core.client.model* a *gemini.core.client.net.model*, které jsou základními kameny všech dalších kroků spjatých s manipulací s torrent souborem.

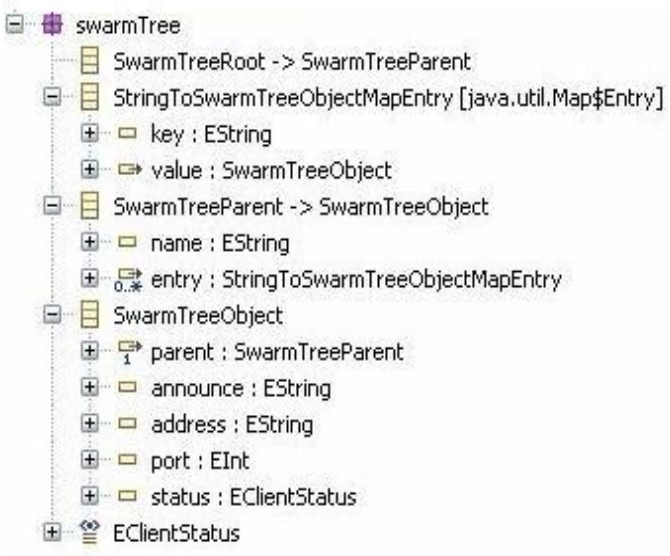
Změny v architektuře modelu v případě, že je využito EMF[6] nejsou dramatické a implementačně náročné. Protože je model oddělen od kontroleru a zobrazení, není třeba pokaždé zasahovat do jiných části kódu. Následující obrázky 22 a 23 ilustruji právě onu vyzdvihovanou vizualizaci.



Obrázek 22 – Diagram tříd: Základní struktura modelu pro zobrazovací komponenty (Generováno pomocí EMF[6] nástrojů)

Vývojář si tedy může vybrat, zda chce modely vytvářet kreslením třídních diagramů, jako na obrázku 22 nebo pomocí editoru z obrázku 23. Při vývoji této aplikace bylo použito druhého způsobu.

Stromová struktura (rodič/potomek) je použitá pro prakticky všechny zobrazení (tabulky i stromy). SwarmTreeRoot je speciální typ rodiče pro nastavení neviditelného kořene v rámci některých vizuálních komponent JFace (TreeViewer). Tímto vizuálním způsobem se modelují EMF[6] modely, ze kterých se následně generuje programový kód. Na obrázku 23 jde konkrétně o nástroj EMF[6] Ecore Editor.



Obrázek 23 - Model Swarm Tree (EMF[6]-Ecore diagram)

V příslušném kontroleru je pak nutné tento model inicializovat. Tato inicializace se provádí při prvním volání modelu s využitím návrhového vzoru Singleton (viz dále).

```
public void createModel() {
    ResourceSetImpl resourceSet = null;
    try {
        resourceSet = new ResourceSetImpl();
        // Register the appropriate resource factory to handle all file
        // extensions.
        resourceSet.getResourceFactoryRegistry().getExtensionToFactoryMap()
                .put(Resource.Factory.Registry.DEFAULT_EXTENSION,
                        new XMIResourceFactoryImpl());

        // Register the package to ensure it is available during loading.
        resourceSet.getPackageRegistry().put(TorrentsPackage.eNS_URI,
                TorrentsPackage.eINSTANCE);

        Resource resource = resourceSet.createResource(torrentURI);
        coreModel = TorrentsFactory.eINSTANCE.createCoreModel();
        resource.getContents().add(coreModel);

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Obrázek 24 - Ukázka kódu: vytvoření modelu (`class ClientModelManager`)

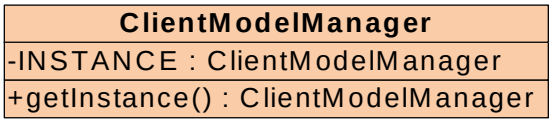
Takto vytvořený EMF[6] model je uložen na disku ve formátu XML(viz slovník pojmů). Na následujícím obrázku je zobrazena situace, kdy byl torrent po nějakou dobu aktivní (byly stahovány data), v tomto modelu dochází ke změnám hodnot a tyto hodnoty jsou ukládány na disk. Oproti obrázku 10 v kapitole 3.1., kde byla zobrazena statická struktura torrentu, zde jsou data z torrentu již importována do modelu, které mění akce kontroleru a která jsou zobrazována na výstupech.

Model torrents.ecore, tak jako většina modelů v projektu, uchovává objekty v mapách (využijeme jedinečných jmen torrentů jako klíče). S mapami se pak výhodně pracuje (je mnohem pohodlnější volat hodnotu mapy než iterovat seznam a testovat nějakou podmínku pokaždé, když chceme něco změnit) a navíc udržují konzistenci projektu tím, že všechny tyto modely pracují nad jedním torrentem, tudíž ke všem přistupujeme stejně napříč funkčními moduly ať se jedná o model torrentu, swarmu, tvorby torrentů apod.

Obrázek 25 – Struktura modelu aktivního torrentu (model torrents.ecore)

Na obrázku je v modelu jeden aktivní torrent, který má dva soubory a čtyři adresy trackerů. Swarm obsahuje jednoho aktivního klienta a dodatečné informace o průběhu stahování obsahují informace o počtech klientů, kteří mají celou kopii (seed) a těch, kteří mají jen část (peer). Dále vidíme, že již bylo staženo 32 KB a celková velikost souborů ke stažení je 264072 bytů.

Návrhový vzor Singleton je v projektu často využíván. Typicky je tato struktura používána pro práci s modely, kdy je nutné přistupovat k instanci modelu – ať je to již prvotní inicializace nebo uložení. Singleton navíc pomocí své statické reference řeší situaci, kdy je potřeba přistupovat k modelu z různých míst kódu bez složitých předávání referencí na objekt.



Obrázek 26 – Diagram tříd: Singleton (`class ClientModelManager`)

```
private static ClientModelManager INSTANCE;

    public static ClientModelManager getInstance() {
        if (INSTANCE == null) {
            synchronized (ClientModelManager.class) {
                INSTANCE = new ClientModelManager();
            }
        }
        return INSTANCE;
    }
```

Obrázek 27 – Ukázka kódu: Singleton (`class ClientModelManager`)

Za zmínku stojí synchronizace, což je užitečné v rámci předcházení vyjímek způsobených kolizí vláken. Tyto výjimky se velmi těžce odlaďují, proto je tento způsob implementace používán pro všechny singletony projektu.

Účelem komponenty *gemini.core.client* je především uchovávat informace o zpracovávaných torrentech, zobrazovat je a manipulovat s nimi. Jedná se tedy o vizualizaci modelu. K propojení modelu a vizuálních prvků slouží Providery (u JFace komponent) :

```
private Image file, dir;
private boolean preserveCase;

...

public String getText(Object arg0) {
      String text = ((File) arg0).getName();
      if (text.length() == 0) {
            text = ((File) arg0).getPath();
      }
      return preserveCase ? text : text.toUpperCase();
}

Image dir, file;
public Image getImage(Object arg0) {
      return ((File) arg0).isDirectory() ? dir : file;
}
```

Obrázek 28 – Ukázka kódu: LabelProvider (`class FileTreeLabelProvider`)

Ukázka kódu na obrázku 28 prezentuje to, co bude zobrazeno u příslušného uzlu ve stromu reprezentující adresářovou strukturu. Text je vracen v podobě jména souboru, pokud není délka názvu nulová nebo cesty k souboru. Co se týče ikony, která je přidána k příslušnému uzlu, pak je v tomto případě (adresářová struktura) situace velmi jednoduchá. Metoda getImage(Object arg0) vrací předem připravený obrázek v závislosti na tom, zda je soubor typu adresář nebo soubor. Pravdivostní hodnota preserveCase pouze upraví text s malými resp. velkými písmeny.

Za zmínku stojí využití základních ikon platformy, které jsou k dispozici voláním statických metod finální třídy PlatformUI.

```
file = PlatformUI.getWorkbench().getSharedImages().getImage(
        ISharedImages.IMG_OBJ_FILE);
```

Obrázek 29 – Ukázka přístupu ke sdíleným obrázkům (**class** `FileTreeLabelProvider`)

Třída PlatformUI je v celém projektu často využívána pro přístup k aplikačnímu oknu a jeho funkcím.

ContentProvider má za úkol zajistit v podstatě zjištění základních informací o struktuře.
V případě vizuální komponenty, která má za úkol zobrazit stromovou strukturu použijeme Content Provider :

```
public Object getParent(Object arg0) {
      return ((File) arg0).getParentFile();
}

public Object[] getChildren(Object arg0) {
      return ((File) arg0).listFiles();
}

public boolean hasChildren (Object arg0) {
      Object [] obj = getChildren (arg0);
      return obj == null ? false: obj.length > 0;
}
```

Obrázek 30 – Ukázka kódu - ContentProvider
(**class** `FileTreeContentProvider`)

Tedy pro každý prvek, který je zpracováván při aktualizaci komponenty a je předáván jako parametr metod zjistí (v tomto případě se jedná opět o adresářový strom) rodičovský uzel, seznam potomků, případně zda uzel vůbec nějaké potomky má.

K tomu, aby byly vizuální komponenty aktuální (synchronizované s modelem) se využívá Adapter. Tyto adaptéry se připojují na EMF[6] objekty takto:

```
public class TorrentTableAdapter extends AdapterImpl implements Adapter {

	private TableViewer tableViewer;
	private TorrentTableContentAdapter contentAdapter;

	public TorrentTableAdapter(TableViewer tableViewer) {
		super();
		this.tableViewer = tableViewer;
		ClientModelManager.getInstance().getCoreModel()
					.eAdapters().add(this);
	}

	@Override
	public void notifyChanged(Notification notification) {
		super.notifyChanged(notification);

		if (notification.getFeature() instanceof EReferenceImpl) {

			EReferenceImpl feature =
				(EReferenceImpl) notification.getFeature();

			if (feature.getName().equals("torrentMap")) {

				switch (notification.getEventType()) {

				case Notification.ADD:
					addTorrentToTableModel(notification);
					break;

				case Notification.REMOVE:
					removeTorrentFromTableModel(notification);
					break;
				}
				refresh();
			}
		}
	}
```

Obrázek 31 – Ukázka kódu: Adapter (`class TorrentTableAdapter`)

V metodě notifyChanged(Notification) ověříme jaký object poslal notifikaci a na základě toho pak z parametru notification můžeme zjistit o jakou operaci jde (přidání, odebrání, nastavení) a zároveň objekt třídy Notification poskytuje informace o nové i staré hodnotě objektu, se kterým je tento adaptér svázán. Pak provedeme aktualizaci vizuální komponenty voláním metody refresh()-popis na další stránce.

Synchronizaci modelu a vizuálních komponent je potřeba věnovat patřičnou pozornost, protože princip vykreslování na obrazovku má svá pravidla, jejichž nedodržení vede k výjimkám typu InvalidThreadAccessException, případně SWTException. V SWT existuje speciální vlákno pro vykreslování grafiky, a my mu předáme příslušné vykreslovací operace vložené do vláken, která předáme jako parametr metody Display.asyncExec(). Operace se tímto způsobem vloží do fronty a provede se, jakmile je to možné (asynchronně). Pokud použijeme metodu Display.syncExec(), přeruší se volající vlákno, které čeká, dokud není operace provedena.

Na následujícím kódu je ilustrováno, jak použít vlákna tohoto typu. Metoda refresh() je součástí adaptéru, který poslouchá na modelu. Pokud je notifikována změna modelu, je po provedení příslušné akce (nastavení, přidání do kolekce atd.) model uložen a komponenta TableViewer je aktualizována.

```
private void refresh() {
	Display.getDefault().asyncExec(new Runnable() {
		@Override
		public void run() {
			TorrentTableModelManager.getInstance().doSave(
					new NullProgressMonitor());

			if (!tableViewer.getControl().isDisposed()) {
				tableViewer.refresh();
			}
		}
	});
}
```

Obrázek 32 – Ukázka kódu: Asynchronní vlákno (`class TorrentTableAdapter`)

Poznámka:
Třída Display reprezentuje propojení mezi platformou, uživatelským rozhraním a SWT.

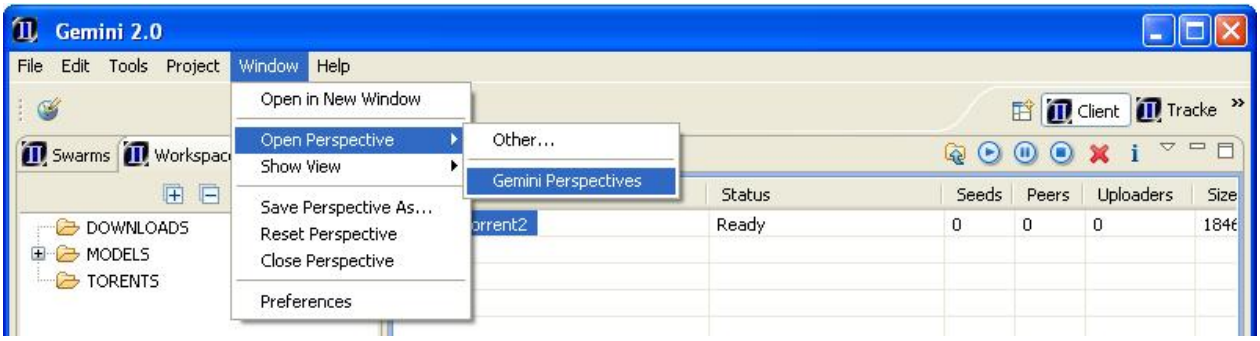
Následující ukázka zdrojového kódu ukazuje, jakým způsobem se přidá JFace komponenta TableViewer do daného kompositu. Za povšimnutí stojí nastavení providerů pro TableViewer, nastavení vstupu TableViewer komponenty – je to EMF[6] mapa objektů !, nastavení adaptérů a nastavení SelectionProvideru.

```
@Override
public void createPartControl(Composite parent) {
	createContents(parent);
	makeActions();
	hookContextMenu();

	new Runnable() {
		public void run() {
			contributeToActionBars();
		}
	}.run();
}

protected void createContents(Composite parent) {
	tableViewer = new TableViewer(parent);
	tableViewer.getControl().setData("TorrentsTableView");

	tableViewer.setContentProvider(new TorrentTableContentProvider());
	tableViewer.setLabelProvider(new TorrentTableLabelProvider());

	tableViewer.setUseHashlookup(true);

	table = tableViewer.getTable();

	initColumns();
	attachListeners();

	inputMap = TorrentTableModelManager
			.getInstance().getTorrentTreeRoot().getEntry();

	tableViewer.setInput(inputMap);

	table.setHeaderVisible(true);
	table.setLinesVisible(true);

	initAdapters(inputMap);

	getSite().setSelectionProvider(tableViewer);
}
```

Obrázek 33 – Ukázka kódu: JFace komponenta - TableViewer
(`class TorrentsTableView`)

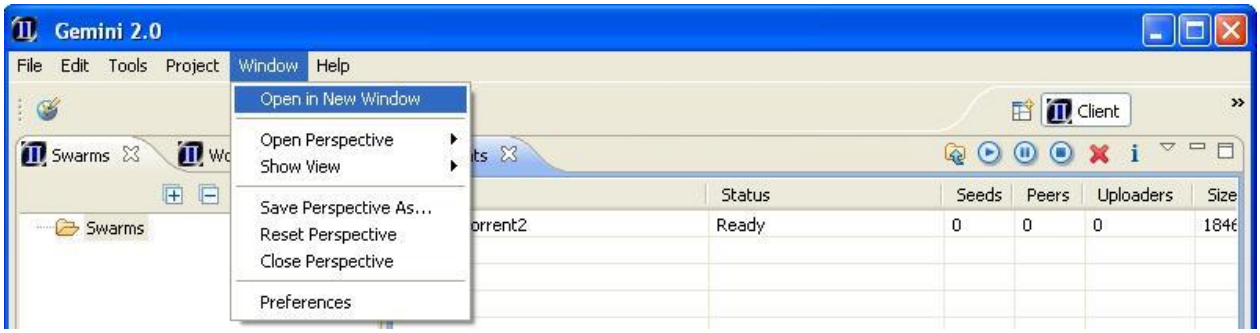
5.3.2. Funkce

Nabídka funkcí z hlavního menu aplikace obsahuje některé společné funkce, které jsou součástí jádra aplikace, tak i další funkce, které přidávají do hlavního menu ostatní pluginy dle svých potřeb. Mezi funkce, které jsou společné, patří především výběry perspektiv, mezi kterými se může uživatel snadno přepínat a nabídka podoken (zobrazení), kterými si může perspektivu přizpůsobit. Tato funkcionalita, spolu s možností měnit pozice a velikosti oken je velkou předností, která dotváří celkově dobrý dojem z uživatelského rozhraní RCP[5] aplikace. Ovládání, ale hlavně práce v takových přizpůsobených, či nově vytvořených perspektivách je intuitivní a přehledná.



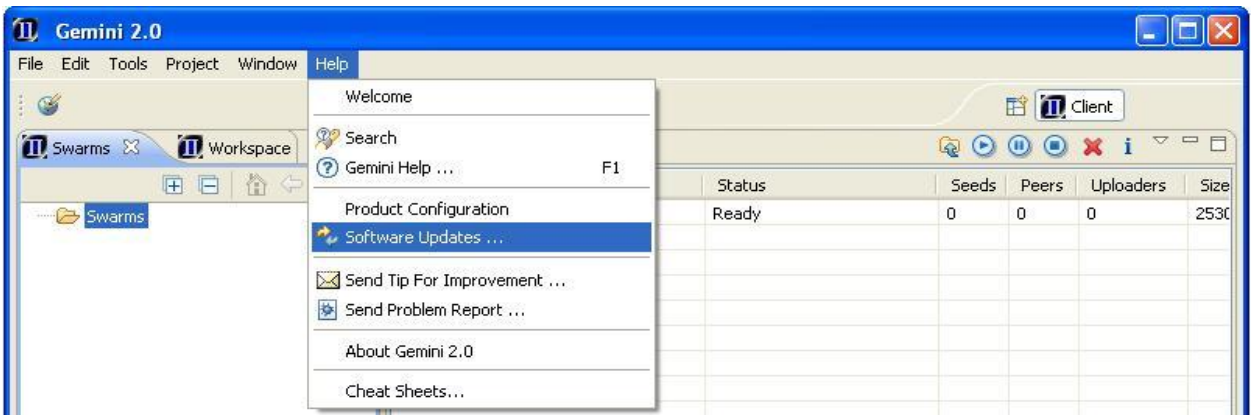
Obrázek 34 – Ukázka funkce: Menu/Window/Open Perspective

Kromě toho, že upravenou perspektivu můžeme uložit nebo vrátit zpět původní nastavení, je možné aplikaci otevřít v novém okně, což může být vhodné pro použití více monitorů nebo pokud chceme kontrolovat více funkčních modulů. Výhodné je to například pro ladění nově vyvíjených komponent, kdy jsou data z těchto komponent zobrazovány v rámci své vlastní perspektivy, ale zároveň mají mít interakci s jinými komponentami, což je u aplikací tohoto typu běžné. Pro příklad uvedu tvorbu torrentů ve funkčním modulu *gemini.core.torrentMaker* s perspektivou Torrent Maker, kdy má uživatel možnost nastavení importu nového torrentu do funkčního modulu *gemini.core.client* s perspektivou Client. Tento torrent se importuje do modelu Finished Torrents, kde čeká na vstup od uživatele k povolení sdílení (a zobrazí se v okně Finished Torrents).

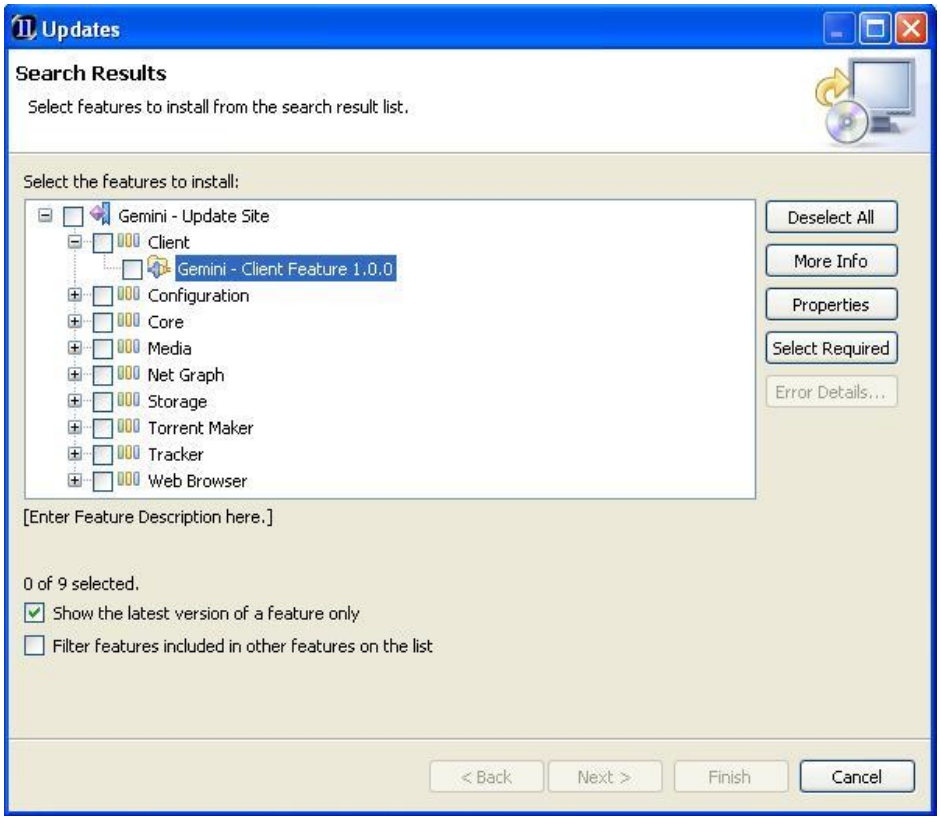


Obrázek 35 – Ukázka funkce: Menu/Window/Open in New Window

Další důležitou společnou funkcí je update. Tato funkce zajistí připojení k nastavenému serveru, kde jsou uloženy aktuální modifikace aplikace.



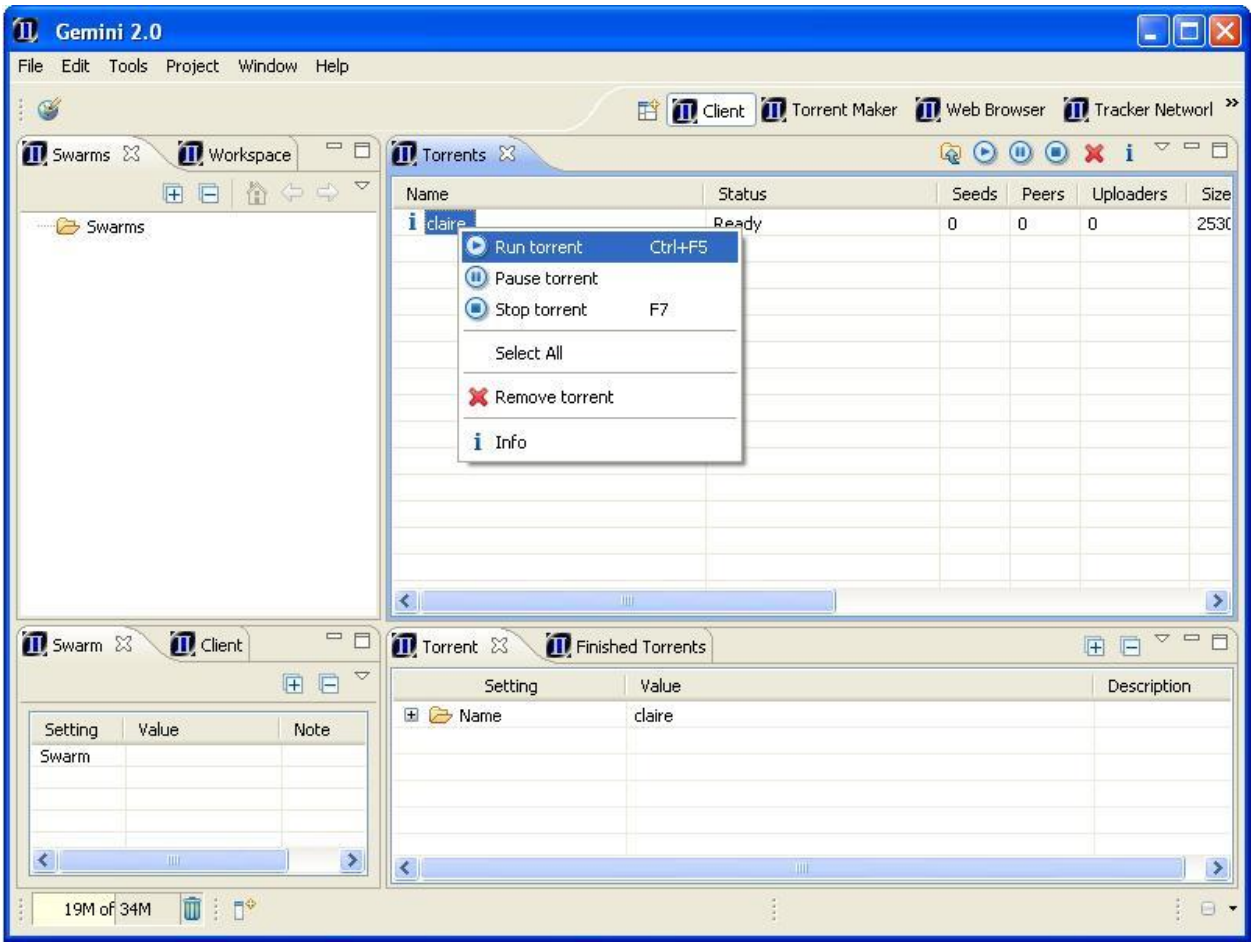
Obrázek 36 – Ukázka funkce: Menu/Help/Software Updates



Obrázek 37 – Ukázka funkce: Software Updates/Wizard

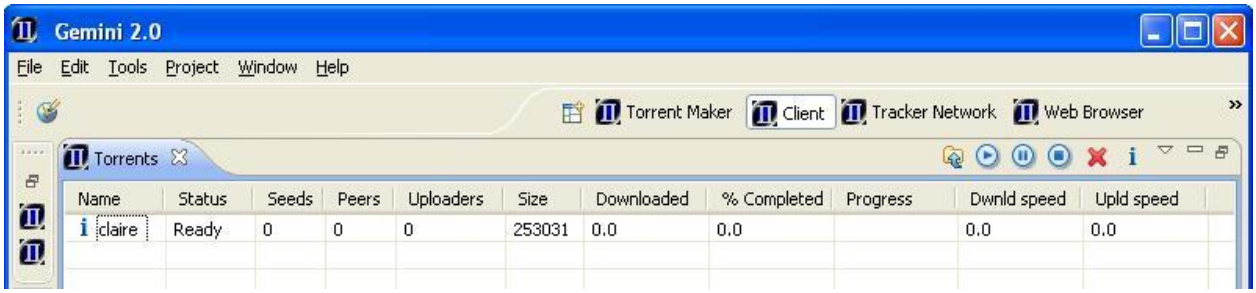
V této fázi je pak možné vybrat funkční moduly pro aktualizaci, pokud na serveru nějaké existují. V rámci těchto aktualizací lze k již obsaženým funkčním modulům přidávat další, pokud byly vydány. Další variantou aktualizace je opravný kód (patch), který pozměňuje původní zdrojový kód. Tyto záplaty mohou obsahovat i více změn najednou, a to i pro různé funkční moduly v rámci aplikace.

Nejdůležitější funkcí klientské aplikace je ovládání stahování torrentu. K tomu slouží rozhraní (perspektiva) Client funkčního modulu *gemini.core.client.net* obsahující seznam stahovaných torrentů. Tato perspektiva je navržena tak, že obsahuje podokna, která mohou být užitečná pro rychlou obsluhu (je samozřejmě možné si tuto perspektivu upravit na míru potřebám uživatele, tak jako u všech ostatních perspektiv). V levé horní části perspektivy jsou k dispozici okna Swarms a Workspace. Okno Swarms poskytuje dynamický stromový pohled na aktivní torrenty. Okno Workspace poskytuje taktéž stromový náhled, a to sice standardní umístění pracovních adresářů Downloads-umístění stahovaných souborů, Models-uložené nastavení oken (nemělo by se s nimi manipulovat) a Torrents-standardní složka pro umístění torrentů.
Dále se v levé dolní části nacházejí okna pro zobrazení detailů z oken v horní části (uživatel si vybere swarm a podrobnosti se zobrazí v tomto okně). Tabulka v pravé horní části zobrazuje torrenty, které jsou stahovány, v dolní části je tabulka se staženými torrenty a dále okna pro detail torrentu. Ovládání je intuitivní buď pomocí pravé klávesy myši nebo z menu.



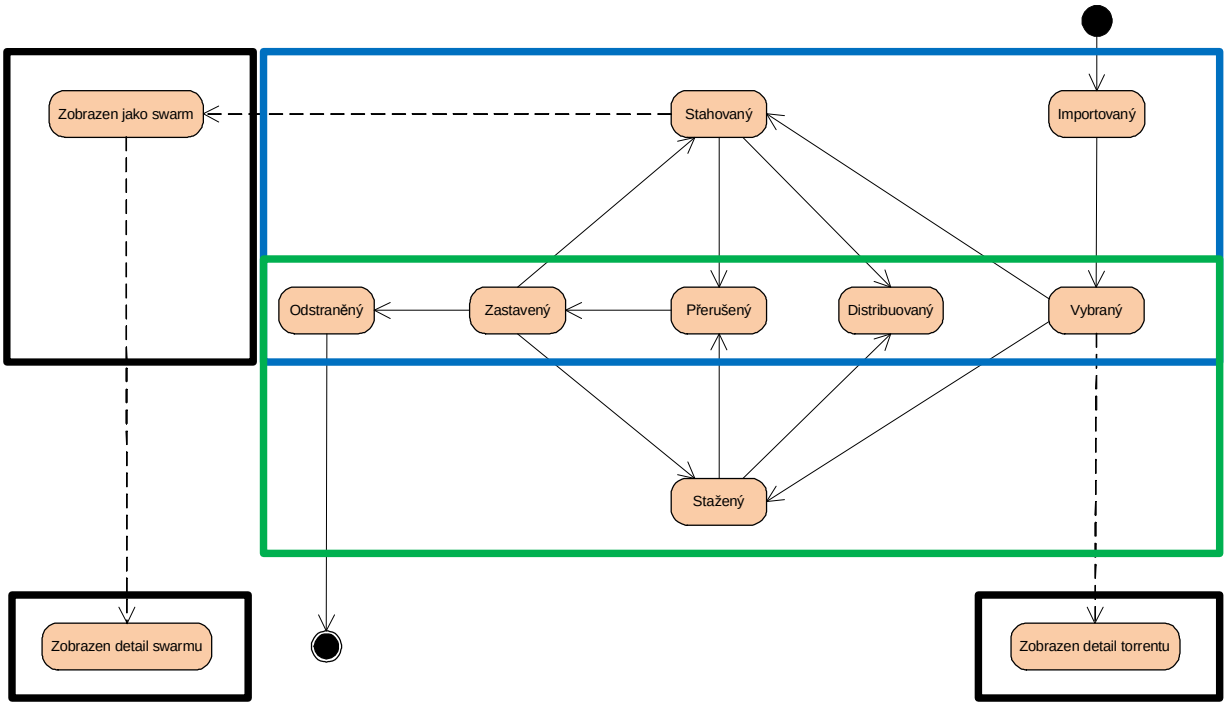
Obrázek 38 – Ukázka funkcí: Ovládání stahování dat (perspektiva Client)

Protože je funkční modul *gemini.core.client.net* nejdůležitějším, na následujícím obrázku jsou zobrazeny všechny sloupce tabulky Torrents, kde jsou zobrazovány všechny podstatné údaje zachycující aktuální stav torrentu. V pořadí zleva doprava je zobrazeno: jméno torrentu, stav, počet seedů, počet peerů, počet klientů, velikost souboru, velikost stažených dat v procentech, sloupce Progress, Dwnld Speed a Upld Speed nejsou v této fázi implementovány.



Obrázek 39 – Detail sloupců tabulky Torrents v perspektivě Client

Následující obrázek je volnějším náhledem zobrazení torrentu a jeho parametrů v rámci perspektivy z obrázku 38 v závislosti na stavu torrentu. Do tabulky Torrents lze torrent importovat, můžeme ho vybrat a spustit stahování. Pokud je torrent aktivní, zobrazí se v levém okně Swarms dynamicky jako swarm. Výběrem jednoho torrentu v tabulkách Torrents a Finished Torrents se podrobnosti o torrentu zobrazí v dolní části perspektivy v okně Torrent. Stahování můžeme pozastavit nebo úplně zastavit, během stahování jsou stažená data automaticky distribuována. Torrent je po stažení přesunut do tabulky Finished Torrents, kde můžeme uplatnit stejné akce, jako při stahování. Torrent je zrušen odebráním.

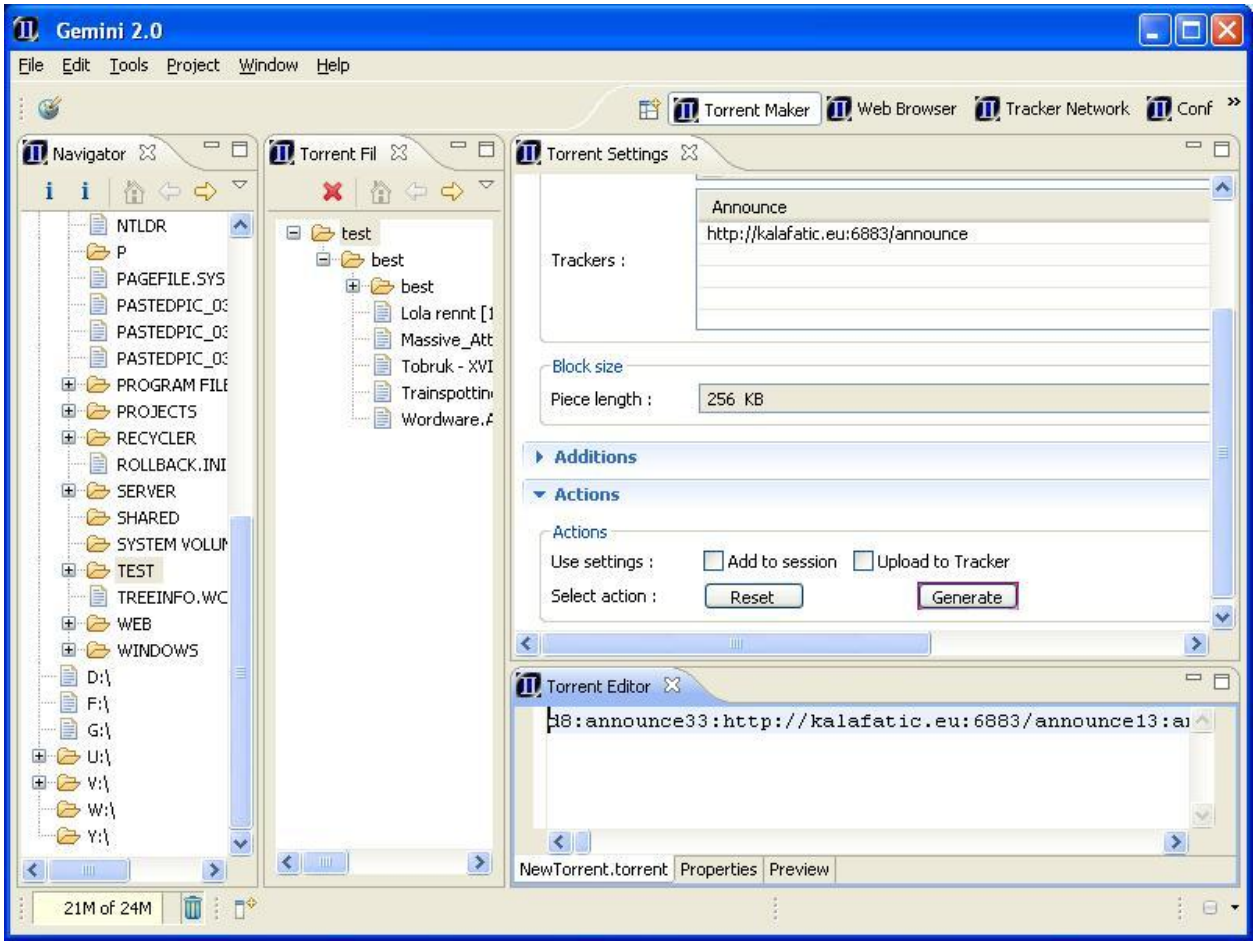


Obrázek 40 – Ukázka zobrazení torrentu v perspektivě Client v závislosti na stavu torrentu

5.4. Rozšíření systému (doplňky)

5.4.1. Tvorba torrentů

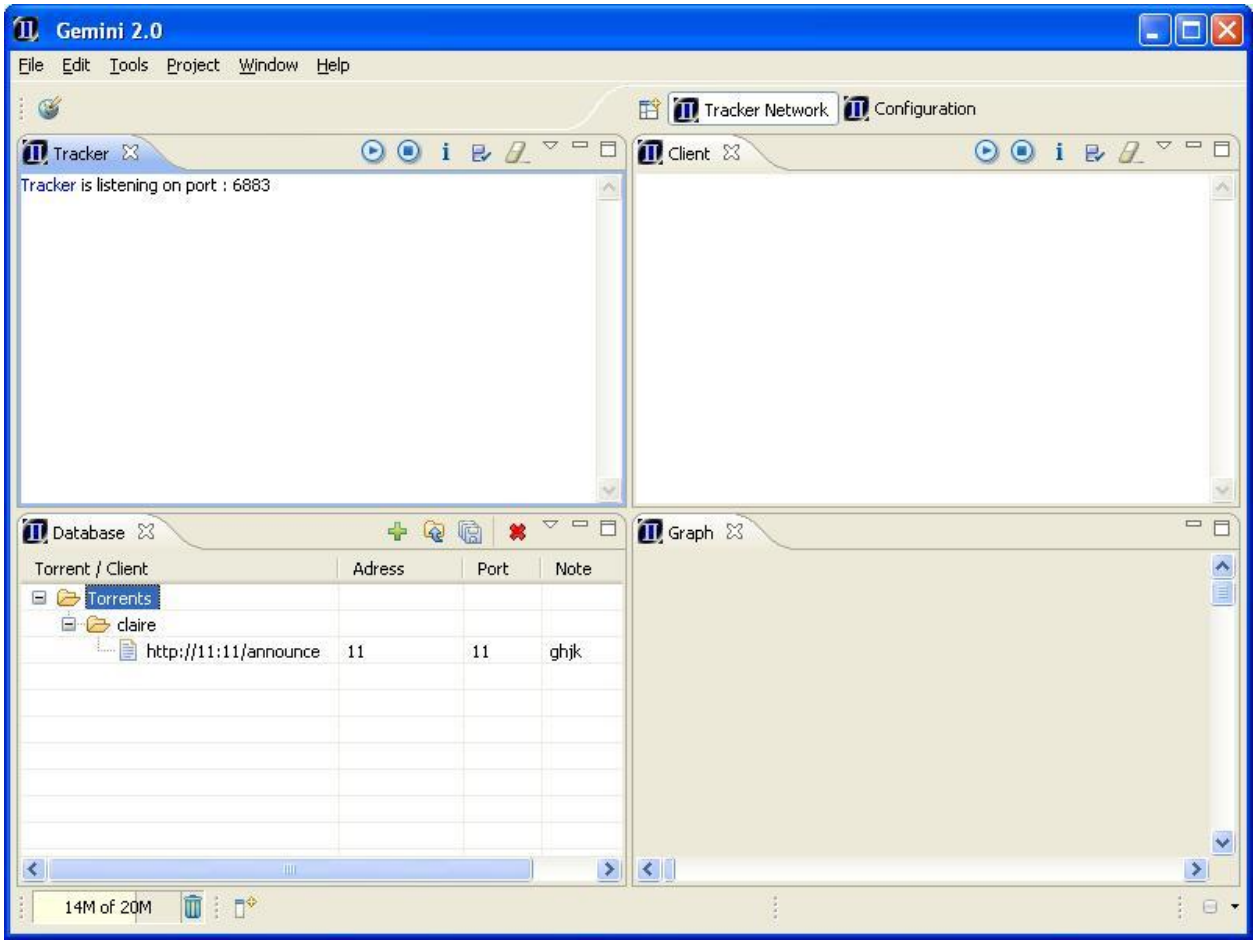
Motivací pro implementaci funkčního modulu Torrent Maker a dále i Trackeru bylo zcela obsáhnout problematiku BitTorrent protokolu. Gemini tak není jen dalším klientem, ale komplexním řešením. Princip tvorby torrentů je jednoduchý – uživatel si vybere libovolný soubor či složku, zadá adresy trackerů (alespoň jednu) a spustí program. Uživatel může nastavit celou řadu parametrů, což je popsáno v BitTorrent specifikaci. Co bych zde zmínil, je spojení funkcí do vzájemně propojeného celku, tedy: uživatel nejen torrent vytvoří, ale měl by mít možnost i umístit torrent na zadané trackery, případně i odeslat ne webový server, který by zpřístupnil tento torrent pro zájemce. Jedna z podporovaných funkcí je i možnost přidání torrentu přímo do klientské aplikace, kde se takto vytvořený torrent chová, jako by byl stažený (můžeme tedy tento torrent umístěný v tabulce Finished Torrents v perspektivě Client ovládat, tedy především distribuovat). Tento přístup by měl zjednodušit a značně zefektivnit stahování dat v rámci BitTorrent sítě.



Obrázek 41 – Ukázka perspektivy: Torrent Maker

5.4.2. Tracker

Účelem trackeru je poskytovat informace, které požadují klienti a dodávat jim pokud možno optimalizovaná data. Tím je myšleno, aby tracker před odesláním odpovědi klientovi provedl analýzu svých dat (v podstatě vyhodnocení statistických údajů, které tracker během své činnosti od klientů získal). Principem je obyčejná serverová aplikace s jednoduchou databází (EMF[6]). Klient po přečtení IP adres trackerů definovaných v torrent souboru tyto dle potřeby kontaktuje a zasílá informace o dotazovaném torrentu a požadavky (dle specifikace komunikace klient/tracker) na adresy klientů sdílejících data daného torrentu. Pro potřeby administrace tohoto serveru je vytvořena kostra trackeru s několika základními funkcemi, které dovolují ovládání serveru, manipulaci s torrenty a sledování klientů ve swarmu.

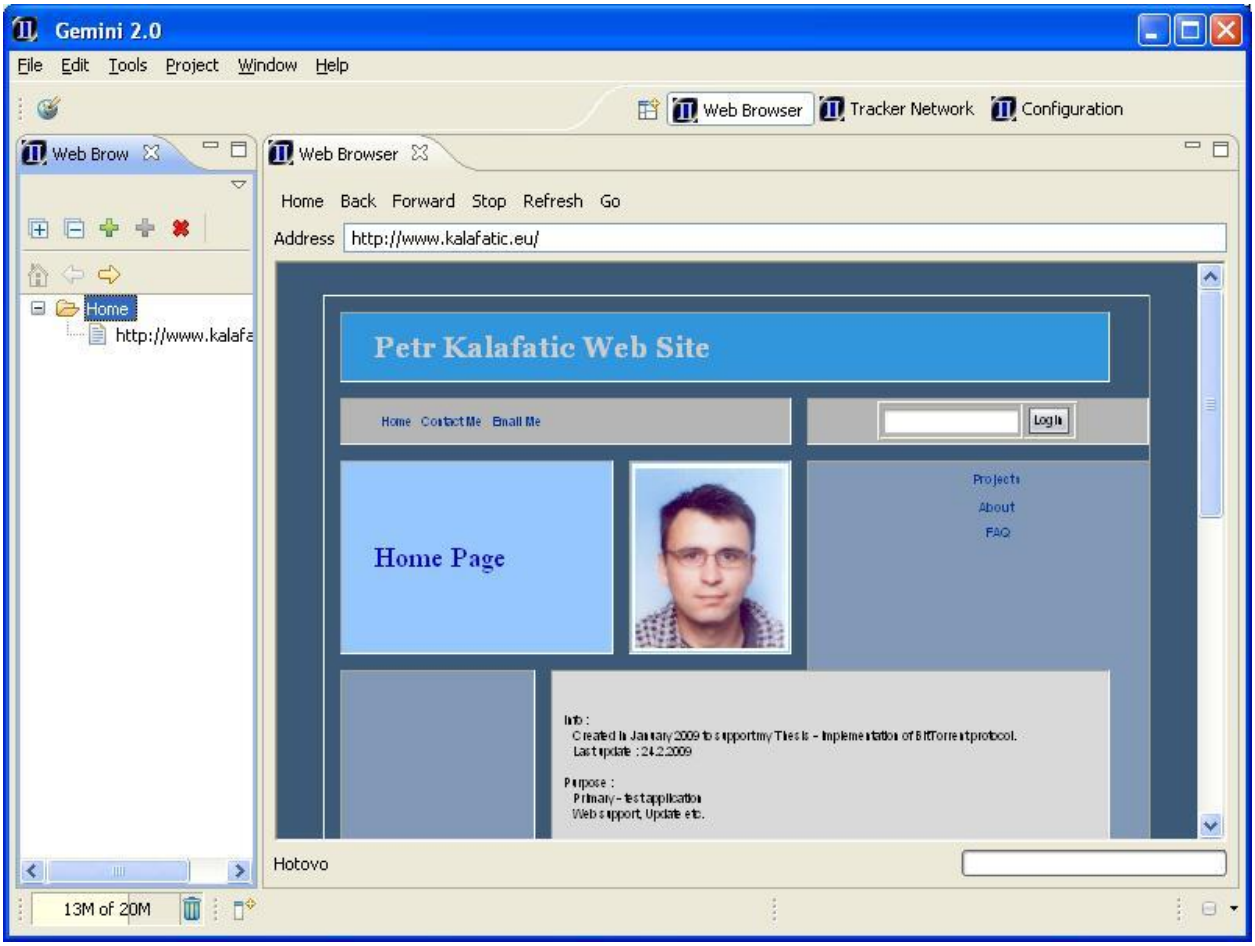


Obrázek 42 – Ukázka perspektivy: Tracker

V levé horní části je umístěna konzole trackeru, kde se zobrazuje stav serveru a akceptovaná připojení jednotlivých klientů. V levé dolní části se zobrazuje databáze torrentů a klientů ve stromové struktuře. V tomto okně můžeme jednoduše tuto databázi spravovat. V pravém horním okně se zobrazují detailní informace, které se zobrazují výběrem objektu v datovém okně. V pravé dolní části perspektivy je zatím neimplementované grafické zobrazení detailů. Doplněk tracker je ve fázi ladění a testování.

5.4.3. Webový prohlížeč

Motivací pro vývoj funkčního modulu webového prohlížeče bylo umožnit uživateli produktu přímo z aplikace vyhledávat, prohlížet a stahovat torrent soubory, čímž se opět zvýší užitná hodnota aplikace a efektivita práce (uživatel nemusí otevírat externí prohlížeč). Implementace je založena na SWT komponentě spojené s modelem tak, aby byla splněna základní funkcionalita webového prohlížeče.



Obrázek 43– Ukázka perspektivy: Web Browser

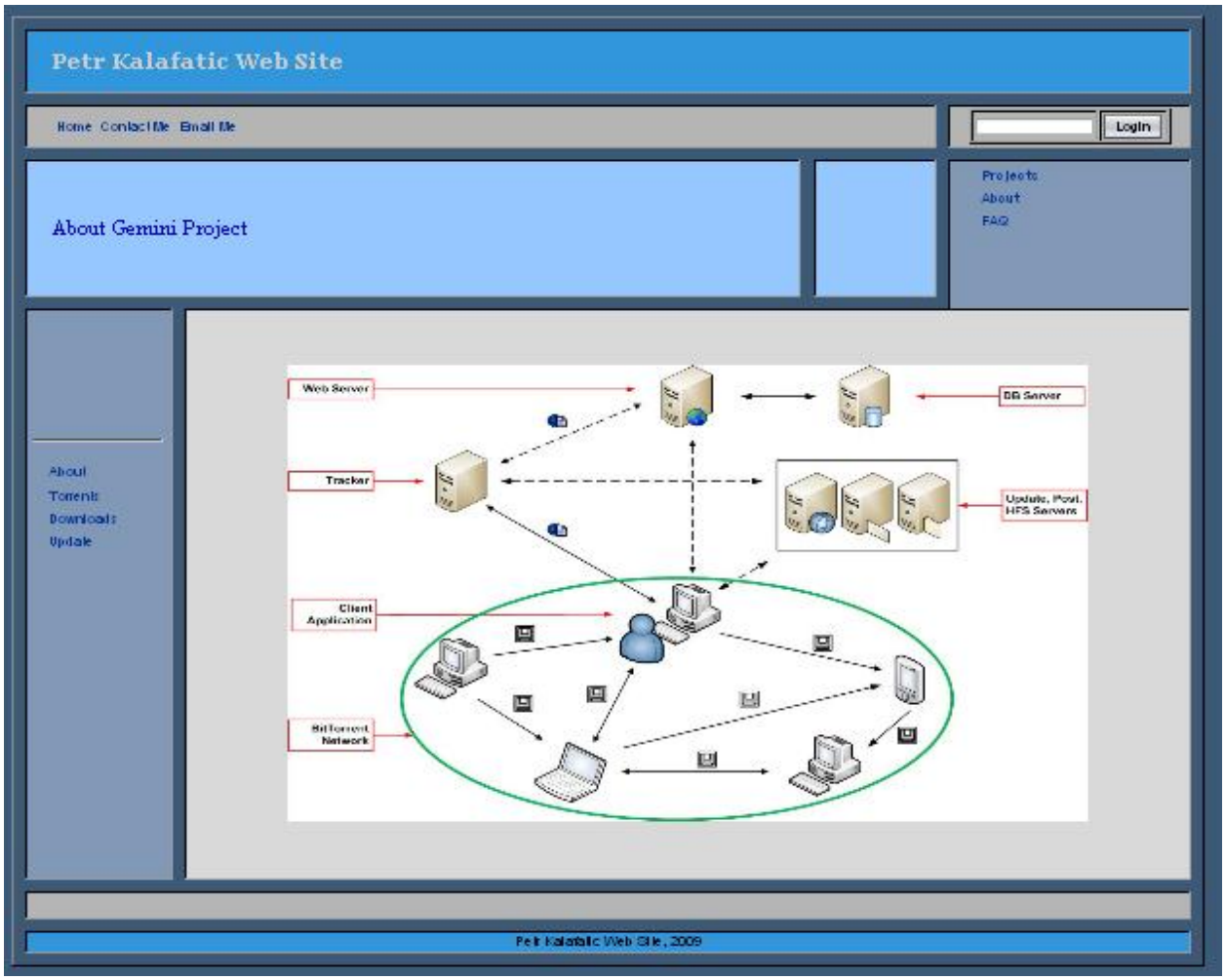
Základními funkcemi doplňku jsou přidávání a odebírání složek a stránek, tak jak to funguje v klasických prohlížečích. Přímo v menu prohlížeče jsou pak tlačítka pro navigaci: Home, Back, Forward, Stop, Refresh, Go a textové pole pro adresu stránky.

Propojení aplikace s webovou aplikací není zcela doladěno, přestože základní implementace je hotova. Toto propojení lze ilustrovat jednoduchou funkcí odeslání vzkazu pro vylepšení nebo ohlášení chyby, kdy prohlížeč zobrazí příslušnou stránku s formulářem, případně otevře externí přednastavený prohlížeč.

5.4.4. Webová podpora

Účelem webové podpory produktu je zajistit úložiště pro torrenty (webová aplikace), prostor pro běh trackeru, aktualizace komponent, kontakt na podporu. Tímto způsobem lze, dle mého názoru, docílit značného zvýšení konkurenceschopnosti produktu.

Správa webové podpory je důležitá i proto, že přináší velmi cenné informace od uživatele (zpětná vazba), o preferencích podobných těm, které byly prováděny ve statistickém průzkumu na počátku vývoje produktu. Je to tedy rozhraní mezi uživateli a vývojáři.

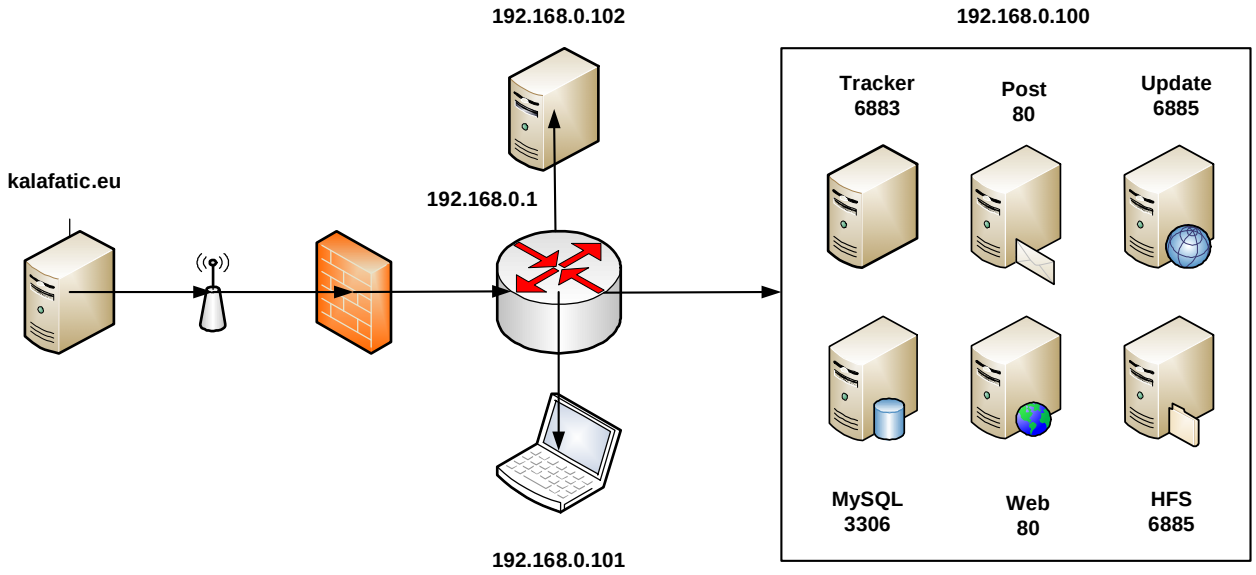


Obrázek 44 – Ukázka stránky projektu

Pro účely této podpory jsem zprovoznil vlastní doménu kalafatic.eu. Pro udržení konzistence projektu jsem použil JSF (Java, NetBeans, MySQL – vše volně použitelné a multiplatformní). Pro běh této podpory je využit webový server Apache Tomcat verze 6.0.18, HFS Http-file-server 2.3.206 beta, poštovní server Post Office, databázový server MySQL verze 5.0.

5.5. Síťová konfigurace projektu

Pro potřeby projektu a hlavně ladění a testování (viz dále), bylo nutné nastavit síťové parametry hardwaru, který byl pro vývoj tohoto projektu použit. Jak již bylo naznačeno v úvodu kapitoly 4, a především demonstrováno obrázkem 17 (Kompletní systém Gemini projektu), obsahuje projekt poměrně hodně síťových prvků, které je nutno nastavit tak aby splňovali požadovanou funkcionalitu. Následující obrázek ilustruje detailněji tuto konfiguraci.



Obrázek 45 – Hardwarová a síťová konfigurace projektu

Jak již bylo zmiňováno v předešlém textu, pro potřeby projektu jsem zřídil doménu kalafatic.eu. Pro tuto doménu bylo potřeba nastavit DNS[18] záznamy (A, MX, CNAME...), konkrétně veřejnou adresu, která se dále od poskytovatele připojení překládá na adresu LAN[18].
Pro potřeby poštovního serveru pak bylo nutné nastavit MX[18] záznam.

Dalším krokem bylo nastavení routeru tak, aby firewall povolil protokoly na daných portech:

FIREWALL RULES LIST

Action	Name	Source	Dest	Protocol,Port
Allow	HTTP	*, *	LAN, 192.168.0.100	*,80
Allow	POP3	*, *	LAN, 192.168.0.100	TCP,110
Allow	SMTP	*, *	LAN, 192.168.0.100	TCP,25
Allow	HTTP	*, *	LAN, 192.168.0.100	*,8080~8090
Allow	HTTPS	*, *	LAN, 192.168.0.100	TCP,443
Allow	HTTP	*, *	LAN, 192.168.0.100	*,6885
Allow	IMAP	*, *	LAN, 192.168.0.100	TCP,143
Allow	HTTP	*, *	LAN, 192.168.0.102	TCP,6883
Allow	DMZ Server	*, *	LAN, 192.168.0.102	*, *

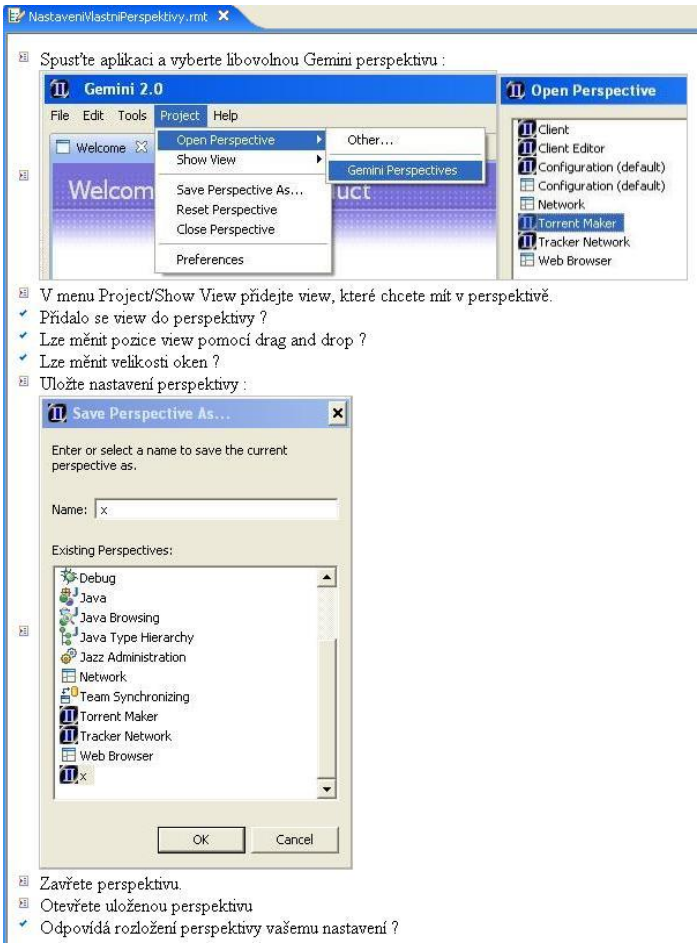
Obrázek 46 – Nastavení pravidel firewallu routeru

6. Testy a porovnání

Testování systému

Testování systému probíhalo v průběhu celého vývoje. Testování je však časově velmi náročné, proto se toto testování blížilo způsobu testování v malých firmách, kde se často pouze u nově vyvinuté funkcionality testuje zadáním očekávaných hodnot a následné kontroly správnosti výsledků. Zhruba v polovině doby vývoje aplikace bylo již testování prováděno v „ostrém“ provozu, kde se testovala hlavně síťová komunikace, která již byla popsána v předchozím textu. Čas nutný k ladění aplikace se opět zvýšil, protože bylo jednak nutné najít vyhovující torrenty (z hlediska velikosti a dostupnosti), dále ověřování typů zpráv posílaných mezi aplikací a trackerem /klientem. Testování doplňků jako je tracker bylo náročné i z pohledu hardwarové konfigurace, protože k dispozici byly jen počítače vzájemně propojené s využitím klasického routeru, jak je ilustrováno na obrázku 45. V poslední fázi vývoje aplikace byl vývoj zastaven (code freeze) a probíhalo jen ladění stávajícího stavu projektu a odstraňování chyb.

Nicméně minimální testování bylo nutné vždy. Následující obrázek demonstruje jednoduchý manuální test (napsán pod IBM Rational Manual Tester).



Obrázek 47 – Manuální test nastavení vlastní perspektivy

CD obsahuje manuální a funkční testy několika podsystémů. Jedná se o testy za použití nástroje Rational od firmy IBM , a ve funkčních testech je ověřována funkčnost například Drag and Drop mezi okny aplikace.

Porovnání s ostatními projekty

Porovnávání bylo jen orientační, detailnější studium jednotlivých BitTorrent aplikací nebylo prováděno. BitTorrent implementací je navíc poměrně hodně, takže jsem porovnal pouze několik, a to sice ty, které jsem využíval. Porovnání je také pouze subjektivní a vychází jednak z cílů této aplikace, a také bere v potaz současný stav projektu.

Na prvním místě bych ohodnotil **Vuze**[16], který byl inspirací tohoto projektu.
Vuze se ovládá intuitivně, poskytuje dostatek informací o stahovaných torrentech, má užitečné funkce, jako například nastavení rychlosti stahování/odesílání dat.

µTorrent: po uživatelské stránce působí velice přívětivě, vyniká především výborným využitím paměti a procesoru

BitTorrent: oficiální jednoduchý klient, zkušenějším uživatelům toho bohužel mnoho nenabízí

BitComet: průměrný klient, možnost komentování torrentu

7. Závěr

Cílem této diplomové práce byla implementace BitTorrent protokolu, která by přinesla do této oblasti datových přenosů něco nového. Po analýze stávajících řešení jsem došel k závěru, že v této oblasti chybí kompletní řešení. Aby bylo možné tohoto cíle dosáhnout, bylo zapotřebí zvolit správné prostředky k dosažení tohoto cíle a adekvátně naplánovat vývoj.

Jak už bylo zmíněno v textu, projekt šel cestou vývoje kompletní podpory BitTorrent protokolu, což nebylo jednoduché z hlediska nastudování a použití různých, často poměrně složitých technologií.

Projekt je v současné fázi kompletním rámcem(kostrou), který je možné dále rozvíjet a vylepšovat.

8. Slovník pojmů

Announce (oznámení)- připojení k trackeru za účelem aktualizace vlastního statusu a získání některých informací, zejména pak aktualizovaného seznamu peerů.
BSD (Berkeley Software Distribution)- BSD je licence pro svobodný software, mezi kterými je jednou z nejsvobodnějších. Umožňuje volné šíření licencovaného obsahu, přičemž vyžaduje pouze uvedení autora a informace o licenci, spolu s upozorněním na zřeknutí se odpovědnosti za dílo.
DHT (Distributed Hash Table)- DHT tracker pracuje podobně jako klasický tracker s announce pro získání seznamu peerů, kteří sdílejí stejný torrent jako vy. Protože je ale DHT tracker distribuovaný či výstižněji řečeno rozptýlený, není zde jen jediný bod, který by v případě havárie ochromil DHT síť. Jestliže se jeden uzel (node) odpojí od DHT, tracker bude pracovat dál. U normálního trackeru znamená výpadek serveru jeho nepoužitelnost. DHT bylo vytvořeno z důvodu jeho odolnosti a nezávislosti na jednom serveru.
Eclipse RCP (Eclipse Rich Client Platform)- Eclipse RCP je open-source projekt, který nabízí alternativní možnost (k samotnému SWT nebo Swingu) jak vyvíjet "vizuálně bohaté" desktopové aplikace v Javě. Představuje minimální množství pluginů (minimální konfiguraci), které je nutné k inicializaci prostředí. Jeho posláním je zjednodušit a zrychlit vývoj komplexních desktopových aplikací, vtisknout jim jednotný a nativní vzhled. Aplikace jsou díky Javě platformě nezávislé a díky SWT mají na každé platformě nativní vzhled. Eclipse RCP je součástí rodiny open-source projektů Eclipse Project, které jsou vyvíjeny pod vedením IBM. Eclipse RCP je samostatnou součástí (podmnožinou) projektu Eclipse Platform, což je velmi známé a úspěšné open-source vývojové prostředí (IDE).
Extension / Extension Point Model-způsob rozšiřování Eclipse RCP je založen na tzv. “extension points”. Pokud chceme přidat do již existující RCP komponenty funkcionalitu, najdeme příslušný extension point, na který “navěsíme” naši akci. Stejně tak můžeme v komponentě definovat extension pointy, které budou moci využít jiní vývojáři k rozšíření funkcionality, aniž by museli zasahovat do kódu naší komponenty.
Gemini (latinsky-dvojčata)-souhvězdí zvěrokruhu, které je umístěno na severní obloze. Toto souhvězdí představuje dvojčata Castora a Polydeuka (Pollux), kteří pluli s Iásonem a argonauty pro zlaté rouno. Významné hvězdy souhvězdí: α Geminorum (Pollux), β Geminorum (Castor), γ Geminorum (Alhena), δ Geminorum (Wasat), ε Geminorum (Mebsuta).
GNU (General Public License)- GNU nebo také GNU GPL (česky „všeobecná veřejná licence GNU“) je licence pro svobodný software, původně napsaná Richardem Stallmanem pro projekt GNU. GPL je nejpopulárnějším a dobře známým příkladem silně copyleftové licence, která vyžaduje, aby byla odvozená díla dostupná pod toutéž licencí. V rámci této filosofie je řečeno, že poskytuje uživatelům počítačového programu práva svobodného softwaru a používá copyleft k zajištění, aby byly tyto svobody ochráněny, i když je dílo změněno nebo k něčemu přidáno. Toto je rozdíl oproti permisivním licencím svobodného softwaru, jejímž typickým případem jsou BSD licence.
HTTP (Hypertext Transfer Protocol) - je internetový protokol určený původně pro výměnu hypertextových dokumentů ve formátu HTML. Používá obvykle port TCP/80, verze 1.1 protokolu je definována v RFC 2616. Tento protokol je spolu s elektronickou poštou tím nejvíce používaným a zasloužil se o obrovský rozmach internetu v posledních letech.

***Hash*-**hashovací funkce je reprodukovatelná metoda pro převod vstupních dat do (relativně) malého čísla, které vytváří jejich jednoznačnou charakteristiku a je velmi obtížné najít taková data, která vyhovují požadovanému otisku (tzv. jednocestná funkce). Výstup hashovací funkce označujeme jako výtah, miniatura, otisk, fingerprint či hash (česky též někdy jako haš). Funkce je důležitou součástí kryptografických systémů pro digitální podpisy a často se též používá pro kontrolu integrity dat, k rychlému porovnání dvojice zpráv, indexování, vyhledávání apod.
HFS-aplikace fungující jako jednoduchý HTTP server, který je určen především pro sdílení souborů.
***IDE (Integrated Development Environment)*-** je software usnadňující práci programátorů, většinou zaměřený na jeden programovací jazyk.Obsahuje editor zdrojového kódu, kompilátor, případně interpret a většinou také debugger.
***IP*-**IP je číslo které jednoznačně identifikuje síťové rozhraní v počítačové síti, která používá IP protokol. V současné době je nejrozšířenější verze IP verze 4, která používá 32bitové adresy zapsané dekadicky po jednotlivých oktetech (osmicích bitů), například `192.168.0.1`. Z důvodu nedostatku IP adres bude nahrazen protokolem IP verze 6, který používá 128bitové IP adresy.
***Leech*-**termín Leech bývá používán pro neslušného peera, který má velmi malý poměr uploadu/downloadu, nebo který opustí swarm hned po tom, co se stane seedem. Leeches obvykle spotřebovávají největší přenosové pásmo swarmu.
***MVC (Model-View-Controller)*-**MVC (někdy označovaná jako Model-2) je softwarová architektura, která rozděluje datový model aplikace, uživatelské rozhraní a řídící logiku do tří nezávislých komponent tak, že modifikace některé z nich má minimální vliv na ostatní části. MVC je často chápán jako návrhový vzor, nicméně se týká architektury aplikací mnohem více než klasický návrhový vzor. Tudíž může být užitečný pojem architektonický vzor (architectural pattern, Buschmann, 1996) nebo možná agregační návrhový vzor (aggregate design pattern).
***OSGi (Open Services Gateway Iniciative)*-**tento nezávislý průmyslový standard definuje komponentovou architekturu. Využívá se především v telekomunikacích a je podporován firmami jako Nokia, Motorola, Philips, atd. Přináší do Javy modularitu. Vlastnosti, kvůli kterým byl integrován do Eclipse RCP jsou zejména pojmenovávání a seskupovaní verzí (tvoření balíčků), management závislostí, rozhraní pro explicitní importy, exporty, updaty, vestavěná bezpečnostní vrstva a dynamičnost
***Peer*-**instance BitTorrent klienta běžícího na počítači. Obvykle je peerem nazýván ten, kdo nemá kompletně stažený torrent.
***P2P (Peer-to-peer)*-**P2P (doslova rovný s rovným), P2P nebo klient-klient je označení architektury počítačových sítí, ve které spolu komunikují přímo jednotliví klienti (uživatelé). Opakem je architektura klient-server, ve které jednotliví klienti komunikují vždy s centrálním serverem či servery, prostřednictvím kterého i komunikují s jinými klienty, pokud je to potřeba. Čistá P2P architektura vůbec pojem server nezná, všechny uzly sítě jsou si rovnocenné (a působí současně jako klienti i servery pro jiné klienty). V praxi se však často pro zjednodušení návrhu v protokolu objevují specializované servery, které ovšem slouží pouze pro počáteční navázání komunikace, „seznámení“ klientů navzájem, popř. jako Proxy server v případě, že spolu z nějakého důvodu nemohou koncové uzly komunikovat přímo. Dnes se označení P2P vztahuje hlavně na výměnné sítě, prostřednictvím kterých si mnoho uživatelů může vyměňovat data. Příkladem takových sítí jsou např. Gnutella či původní verze Napsteru.
Jednou ze základních výhod P2P sítí je fakt, že s rostoucím množstvím uživatelů celková dostupná přenosová kapacita roste, zatímco u modelu klient-server se musí uživatelé dělit o konstantní kapacitu serveru, takže při nárůstu uživatelů klesá průměrná přenosová rychlost.

***Reseed*-**pokud má jeden nebo více leechů „roztažený“ nějaký torrent a není již k dispozici celý tj. několik leechů má několik různých částí, ze kterých ale nelze sestavit kompletní soubor, nebo torrent (první znak umírajícího torrentu, to se nestane, pokud každý má ratio alespoň 1), mohou se pokusit poprosit někoho, kdo ho má celý aby začal znovu seedovat a torrent opět oživil tj. požádat o Reseed.
***RUP (Rational Unified Process)*-**RUP je rozsáhlá, propracovaná objektově orientovaná iterativní metodologie vývoje softwaru. RUP náleží do skupiny. tzv. přístupů řízených případy použití (use-case-driven approach), což znamená, že jako základní element je chápán případ použití definovaný jako posloupnost akcí prováděných systémem či uvnitř systému, která poskytuje určitou hodnotu uživateli systému (v nejobecnějším smyslu).
***Seed*-**peer, který má kompletní kopii torrentu a stále nabízí upload. Čím více seedů je ve swarmu, tím větší bývá rychlost downloadu a také se zvyšuje šance na stažení kompletního souboru. Seedováním je torrent udržován v chodu.
***Soubor .torrent*-**obsahuje metadata o distribuovaných souborech. Obsahuje jména souborů, jejich velikosti a kontrolní součet (viz Hašovací funkce) jednotlivých bloků torrentu. Také obsahuje adresu trackeru (většinou PHP skript).
***Swarm*-**Všichni peers, kteří sdílí torrent, se nazývají swarm. Například šest leeches a jeden seed je swarm (svazek) sedmi.
***SWT (Standard Widget Toolkit)*-**SWT je základní grafická knihovna po Javu, která nemá žádnou závislost na standardním grafickém API Javy. Jeho výkladní skříň je vývojové prostředí Eclipse a je IBM iniciativou. Je sice napsáno v Javě, ale nesmí používat ochranou známku Java IDE. Proč? Protože není Java pure. To znamená, že v sobě obsahuje nativní kód z jednotlivých operačních systémů, na které je portováno. SWT lze s určitou nadsázkou považovat za knihovnu, která leží mezi AWT a Swingem. Komponenty jsou "vybavenější" než u AWT, ale nemají takové množství vlastností jako u Swingu.
***Torrent*-**je buď soubor .torrent, tedy soubor metadat o downloadu, nebo všechny soubory, které jsou jím popisovány.
***Tracker*-**je služba, která zprostředkovává a režíruje spojení mezi klienty (přechovává seznamy IP adres peerů), ale data přes něj netečou, ani nemá žádnou kopii torrentu. (Při sdílení musíte nastavit adresu jejího rozhraní pro nabízení torrentů tj. nejčastěji „adresa/announce.php“ a následně nahrát .torrent na tracker - to většinou vyžaduje registraci a přihlášení pod vaším jménem).
***XML* (*eXtensible Markup Language)*-** je obecný značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C. Umožňuje snadné vytváření konkrétních značkovacích jazyků pro různé účely a široké spektrum různých typů dat.Jazyk je určen především pro výměnu dat mezi aplikacemi a pro publikování dokumentů. Jazyk umožňuje popsat strukturu dokumentu z hlediska věcného obsahu jednotlivých částí, nezabývá se sám o sobě vzhledem dokumentu nebo jeho částí. Prezentace dokumentu (vzhled) se potom definuje připojeným stylem. Další možností je pomocí různých stylů provést transformaci do jiného typu dokumentu, nebo do jiné struktury XML.

9. Literatura

[1] Wikipedia
http://cs.wikipedia.org, http://en.wikipedia.org

[2] Jim D´Anjou, Scott Fairbrother, Dan Kehn, John Kellerman, Pat McCarthy – Eclipse Second Edition

[3] Eric Clayberg, Dan Rubel – Eclipse, Building Commercial-Quality Plug-ins

[4] Frank Budinsky, David Steirberg, Ed Merks, Raymond Ellersick, Timothy J. Grose - Eclipse, Modeling Framework

[5] Eclipse RCP
http://wiki.eclipse.org

[6] Eclipse EMF
http://www.eclipse.org/modeling/emf/

[7] Jakub Maléř, Eclipse RCP – Semestrální projekt IT
http://nb.vse.cz/~zelenyj/it380/eseje/xmalj35/EclipseRCP.htm

[8] Swing versus SWT
http://tomas-net.blogspot.com/2004/05/swing-versus-swt.html

[9] SWT
http://www.eclipse.org/swt/
http://en.wikipedia.org/wiki/Standard_Widget_Toolkit

[10] P2P exPerience - Vše o P2P sítích
http://www.p2pxp.info/index.php?page=exeem

[11] BitTorrent - specifikace
http://wiki.theory.org/BitTorrentSpecification

[12] Bram Cohen- domovská stránka
http://bitconjurer.org/

[13] PC Tunning - BitTorrent jak ho neznáte
http://pctuning.tyden.cz/index.php?option=com_content&task=view&id=8682&Itemid=96

[14] Ing. Marek Běhálek - RUP
http://www.cs.vsb.cz/behalek/frvs/2005/rup/technologie_rup.pdf

[15] Vladimír Rytíř – RUP and small projects
http://www.feec.vutbr.cz/EEICT/2008/sbornik/02-Magisterske%20projekty/07-Informacni%20systemy/06-xrytir01.pdf

[16] Vuze - domovská stránka
http://www.vuze.com/Index.html

[17] Jim Arlow, Ila Neustadt - UML2 a unifikovaný proces vývoje aplikací

[18] Debra Littlejohn Schinder – Počítačové sítě

10. Přílohy

10.1. Obsah CD

- BitTorrent Specification (oficiální specifikace BitTorrent protokolu)
- Build (obsahuje spustitelný soubor aplikace)
- DB – MySQL (exportovaná databáze)
- Design Documents (návrhy projektu)
- Programmer Documentation (javadoc)
- Statistics (statistický průzkum a jeho vyhodnocení)
- Tests (manuální a funkční testy IBM Rational)
- User Documentation (uživatelská dokumentace)
- Workspace App – Eclipse (zdrojové kódy aplikace)
- Workspace Web – NetBeans (zdrojové kódy webové aplikace)